**Informatik-Bericht Nr. 2009-06**

# Umsetzung eines modellbasierten durchgängigen Entwicklungsprozesses für AUTOSAR-Systeme mit integrierter Qualitätssicherung

## A comprehensive Description of a Model-based, continuous Development Process for AUTOSAR Systems with integrated Quality Assurance

Tobias Carsten Müller[1] Malte Lochau[2]
Stefan Detering[3] Falko Saust[1] Henning Garbers[1]
Lukas Märtin[2] Thomas Form[1] Ursula Goltz[2]

Dezember 2009

[1]Institut für Regelungstechnik, TU Braunschweig
[2]Institut für Programmierung und Reaktive Systeme, TU Braunschweig
[3]Institut für Verkehrssicherheit und Automatisierungstechnik, TU Braunschweig

Veröffentlicht: 16.12.2009
http://www.digibib.tu-bs.de/?docid=00031645

## Zusammenfassung

Der AUTOSAR-Standard definiert neben einer durchgängig werkzeuggestützten und modellbasierten Methodik zur Entwicklung von Steuergeräte-Software eine technische Infrastruktur als standardisierte Steuergeräte-Basissoftware zur Implementierung dieser Systeme im Automobil. Die wesentlichen Herausforderungen in der Entwicklung automotiver Systeme ergeben sich dabei nicht nur aus der stetig steigenden Menge korrekt umzusetzender Funktionalität, sondern auch aus der wachsenden Anzahl zusätzlich zu erfüllender Qualitätsanforderungen, wie z.B. Sicherheit, Performanz oder Kosten. Die Integration von Ansätzen zur frühzeitigen, Entwicklungsphasen begleitenden Überprüfung von Korrektheits- und Qualitätskriterien kann dabei maßgeblich zur Beherrschbarkeit der Komplexität dieser Systeme beitragen. Es wird ein entsprechend durchgängig werkzeuggestützter und modellbasierter Entwicklungsprozess, basierend auf dem V-Modell sowie dessen Integration in die AUTOSAR-Methodik definiert. Neben der Überprüfung der funktionalen Korrektheit durch systematische Testverfahren sieht das erweiterte Prozessmodell die Bewertung beliebiger Qualitätskriterien für das zu entwickelnde System vor. Es wird beschrieben, wie insbesondere im AUTOSAR-Kontext der Entwurf der Systemarchitektur die hierfür entscheidende Design-Phase darstellt und als Grundlage für Qualitätsabschätzungen durch Architektur-Evaluation dienen kann. Die Vorgehensweise in den einzelnen Entwicklungsschritten wird detailliert anhand einer umfangreichen, vollständig AUTOSAR-konformen Fallstudie, bestehend aus einem vereinfachten PKW-Komfortsystem, demonstriert. Die durchgängige Toolkette umfasst alle Phasen von der Anforderungsspezifikation bis zur Implementierung auf einem prototypischen Hardware-Demonstrator bestehend aus vier über CAN vernetzten Steuergeräten und HIL-Schnittstellen für die Testdurchführung. Es wird auf ausgewählte Implementierungsdetails, notwendige Workarounds und Besonderheiten der prototypischen Umsetzung eingegangen und die Konfiguration des ausgewählten AUTOSAR-konformen Betriebssystems beschrieben. Abschließend werden die Beobachtungen über Stärken und Schwächen in Bezug zum Potential der AUTOSAR-Initiative gesetzt und diskutiert.



**Stichwörter** Automotive Fallstudie, AUTOSAR, Architektur-Evaluation, modellbasierter Entwicklungsprozess, automatische Codegenerierung, Toolkette, HIL

## Abstract

The AUTOSAR standard defines a seamless tool supported and model based methodology for ECU software design and engineering. Furthermore, the standard specifies a technical infrastructure by means of standardized basic software modules for ECU networks, serving as a uniform implementation platform for AUTOSAR systems. The major challenges in automotive systems development not only arise as a result of the contiuously growing amount of functionality to be realized correctly, but also from the increasing number of quality requirements to be taken into account, e.g. safety, performance, and costs. The integration of approaches for early checking of correctness and quality criteria accompanying the different development phases makes a significant contribution towards coping with the complexity of such systems. We describe such a model based development process and a corresponding tool chain based on the V-modell and its embedding into the AUTOSAR methodology. For the validation of functional correctness systematic testing approaches are applied, and for quality criteria according evaluation methods are used. We discuss that especially in the context of AUTOSAR, the phase of architectural system design is crucial for the quality properties of the system under development, and to what extent architecture evaluation can be used for quality estimation. The practices in the different development steps are illustrated in detail by means of a comprehensive, AUTOSAR compliant case study, i.e. a body comfort system. The tool chain proposed comprises all development stages, starting from the requirements specification, and concluding with the system implementation on a hardware demonstrator prototype. The demonstrator consists of ECUs coupled via CAN, as well as HIL interfaces for test case applications. We give detailled insights in selected implementation issues, workarounds required, and the configuration steps needed for the AUTOSAR operating system. A discussion of the pro's and con's regarding the potential of AUTOSAR concludes.

**Keywords** automotive case study, AUTOSAR, architecture evaluation, model based development prozess, automated code generation, tool chain, HIL

# Inhaltsverzeichnis

# 1 Einführung

Steuergerätenetzwerke moderner Automobile sind heute zunehmend komplexe, eingebettete Software-Systeme [SZ06]. Die Komplexität ergibt sich nicht nur aus der reinen Anzahl zu realisierender Kundenfunktionen und der hierfür verwendeten großen Menge Steuergeräte, sondern ebenso aus dem hohen Vernetzungsgrad der Einzelfunktionen untereinander. Immer neue Ausstattungsmerkmale, eine Vielzahl möglicher Varianten in modernen Fahrzeugprojekten sowie die Implementierung auf heterogenen Ablauf- und Kommunikationsplattformen mit spezialisierter Sensorik/Aktorik, Bussystemen etc. stellen die aktuellen Herausforderungen in der Entwicklung dar. Neben der korrekten Umsetzung und Implementierung der geforderten Gesamtfunktionalität sollen gleichzeitig übergeordnete Qualitätsanforderungen, wie zum Beispiel Kosten und Gewicht oder auch Sicherheit und Modifizierbarkeit erfüllt werden.

Mit dem Ziel die Qualität, Kosten und Beherrschbarkeit künftiger automotiver Systeme zu verbessern wurde von namhaften Hersteller und Zulieferer die AUTOSAR-Initiative [AUT06] gegründet um so einen gemeinsamen ganzheitlichen Standard für die Spezifikation und Implementierung von Steuergeräte-Software zu etablieren. Darüber hinaus definiert der Standard eine strukturierte Entwurfs-Methodik sowie ein Metamodell [KF08] zur Spezifikation von AUTOSAR-Systemen.

## 1.1 Hintergrund und Motivation

Um trotz der hohen Anzahl verschiedener Funktionalitäten, deren Abhängigkeiten untereinander und die daraus resultierende Komplexität eines Fahrzeugsystems handhabbar zu halten ist ein in allen Phasen strukturierter Entwicklungsprozesses wesentlich für den Erfolg neuer Fahrzeugprojekte. Neben einer vollständigen und korrekten Anforderungsdefinition stellt vor allem der Architekturentwurf einen entscheidenden Schritt dar. Frühzeitige Überlegungen zur Auswahl, Anordnung und Verbindung grundlegender Komponenten für die nachfolgende Realisierung der Gesamtfunktionalität erlauben nicht nur die Beherrschbarkeit des Entwurfs durch systematische Dekomposition, sondern auch frühzeitige Aussagen über Eigenschaften des Systems, die über das rein Funktionale hinausgehen. Diese nicht-funktionalen bzw. extra-funktionalen Eigenschaften können im Allgemeinen unter dem Begriff Qualitätskriterium zusammengefasst werden. Diese sind zwar in der Regel entkoppelt von spezifischen Fahrzeugfunktionen, müssen

aber ebenfalls als Teil des Anforderungskatalogs für das zu entwickelnde System berücksichtigt werden, um zu einer akzeptablen Systemimplementierung zu gelangen.

Die zunehmende Bedeutung, die Qualitätskriterien für Entwurfsentscheidungen neuer Systeme haben, wird exemplarisch nicht nur an dem Kriterium Kosten deutlich, das den treibenden Faktor, insbesondere im Bereich der Massenproduktion darstellt. Vielmehr spielen vermehrt auch die Einhaltung der Sicherheitsanforderungen, nicht zuletzt von Seiten des Gesetzgebers wie z.B. durch SIL (Sicherheits-Integritätslevel, IEC 61508/IEC61511) gefordert, bereits im Rahmen des Architekturentwurfs eine entscheidende Rolle für die Qualität eines Systems. Entsprechende Standardisierungsmaßnahmen (z.B. ISO/IEC 9126) für alle maßgeblichen Qualitätskriterien, insbesondere für Software-Systeme, machen ebenfalls die wachsende Bedeutung dieser Faktoren deutlich. Neben der Schwierigkeit, diese Art von Kriterien mit Bezug auf ein konkretes System geeignet zu charakterisieren, besteht die grundlegende Problematik dann vor allem darin, dass etablierte Techniken zur Sicherung der Systemqualität auf funktionaler Seite (Fehlerfreiheit), wie beispielsweise Testen oder Simulation, hierfür in der Regel nicht adäquat sind. In allen anderen Fällen würden mithilfe dieser Techniken erkannte Fehler, wie zum Beispiel Mängel in der Verfügbarkeit einer bestimmten Funktionalität erst sehr spät im Entwicklungsprozess erkannt werden. Da die notwendigen Korrekturen durch die große Anzahl betroffener Systemartefakte mit einem enorm hohen Aufwand verbunden wären, würde dies zu hohem Zusatzaufwand und entsprechenden Kosten führen.

Wir beschreiben eine umfangreiche Fallstudie eines auf dem V-Modell aufbauenden modellbasierten Entwicklungsprozesses von der Anforderungsbeschreibung bis hin zur Implementierung, Integration und Test von Steuergeräten sowie dessen Integration in die AUTOSAR-Methodik. Dabei wird nicht nur die verwendete Toolkette sondern auch eine Methodik zur Überprüfung der funktionalen Korrektheit sowie zur Definition und Auswertung beliebiger Qualitätskriterien für das zu entwickelnde System vorgestellt. Die Vorgehensweise in den einzelnen Entwicklungsschritten wird detailliert anhand einer umfangreichen, vollständig AUTOSAR-konformen Fallstudie, bestehend aus einem vereinfachten PKW-Komfortsystem, demonstriert. Die durchgängige Toolkette umfasst alle Phasen von der Anforderungsspezifikation bis zur Implementierung auf einem prototypischen Hardware-Demonstrator, bestehend aus vier über CAN vernetzten Steuergeräten und HIL-Schnittstellen für die Testdurchführung. Wir gehen auf ausgewählte Implementierungsdetails, notwendige Workarounds und Besonderheiten der prototypischen Umsetzung ein und beschreiben die Konfiguration des ausgewählten AUTOSAR-konformen Betriebssystems. Abschließend werden die Beobachtungen über Stärken und Schwächen in Bezug zum Potential der AUTOSAR-Initiative gesetzt und diskutiert.

## 1.2 Quellen und verwandte Arbeiten

Wir geben zunächst einen kurzen Überblick über die Quellen und weiterführende Literatur, auf die sich die verschiedenen Themenbereiche dieses Beitrags beziehen. Eine grundlegende Einführung in die allgemeinen Themenbereiche des automotiven Software-Engineering gibt [SZ06]. Grundlagen strukturierte Entwicklungsprozesse wie das in diesem Beitrag verwendete V-Modell sind im Detail in [Bal98] beschrieben.

Einen guten Überblick über die Grundlagen und Werkzeuge des Anforderungsmanagements gibt [Poh08].

Einen übersichtlichen Einblick in den AUTOSAR Standard bietet neben den Spezifikationsdokumenten [AUT07a] unter anderem [KF08].

Eine ausführliche Beschreibung der Konzepte des Architekturentwurfs und der Qualitätsbewertung durch Architektur-Evaluation liefern [BCK03]. Beispiele für Qualitätsstandards sind unter anderem ISO/IEC 9126[1] bzw. DIN 66272 für Software-Qualität, der Sicherheits-Integritätslevel (SIL)[2] oder auch ISO 26262 für sicherheitsrelevante elektrische/elektronische Systeme in Kraftfahrzeugen. Der Ansatz zur strukturierten Architektur-Evaluation, auf den wir uns in diesem Beitrag beziehen, basiert auf [FGB+07].

Die in diesem Beitrag beschriebenen Ansätze zur Korrektsüberprüfung als Teil der Qualitätssicherung im rechten Ast des V-Modells, wie zum Beispiel automatisierte Tests und modellbasiertes Testen, sind ausführlich in [Lig02] beschrieben.

Weitere Quellen sind nachfolgend den entsprechenden Kapiteln zu entnehmen.

Einige Teile des Bereichts wurden bereits in [Gar09] und [Mic09] veröffentlicht.

## 1.3 Gliederung

In diesem Bericht gehen wir zunächst in Kapitel 2 auf den AUTOSAR-Standard sowie einige die Implementierung betreffende Besonderheiten in Kapitel 4.4 ein. In Kapitel 3 stellen wir die Fallstudie, beginnend mit dem Demonstrator und der Funktionalitätsbeschreibung vor und gehen anschließend auf das verwendete Prozessmodell, den verwendeten Werkzeugverbund und einigen Anmerkungen zu Korrektheits- und Qualitätsanforderungen näher ein. In Kapitel 4 wird auf die einzelnen Schritte des in Kapitel 3.3 beschriebenen Prozessmodells ausführlich eingegangen. Wir schließen in Kapitel 5 mit einer Diskussion der beobachteten Stärken und Schwächen sowie Potentiale des AUTOSAR-Standards für die Entwicklung künftiger Steuergeräte-Software.

[1] http://www.iso.org/

[2] siehe auch IEC 61508/IEC61511

# 2 Der AUTOSAR-Standard

Die AUTOSAR-Initiative als ein Zusammenschluss namhafter Hersteller und Zulieferer automotiver Systeme hat sich zum Ziel gesetzt, gemeinsame Rahmenbedingungen für die Entwicklung und den Austausch sowohl von Software- als auch Hardware-Komponenten derartiger Systeme zu schaffen. Der AUTOSAR-Standard definiert nicht nur eine umfassende technische Infrastruktur für diese Systeme, sondern auch eine darauf aufbauende Methodik und Beschreibungsformate für die Entwicklung AUTOSAR-konformer Systemspezifikationen. Auf dieser Grundlage kann die Etablierung durchgängig werkzeuggestützter Entwicklungsprozesse erfolgen.

Zum AUTOSAR-Standard gehört somit zum Einen die Standardisierung sämtlicher Schnittstellen einer vollständigen technischen Infrastruktur für Steuergeräte-Plattformen. Zum Anderen wird ein Metamodell propagiert, mit dem alle wesentlichen Bestandteile einer AUTOSAR-konform beschriebenen Systemspezifikationen in einem einheitlichen Datenmodell erfasst werden. Die AUTOSAR-Methodik definiert anhand entsprechender Sichten auf Teilaspekte dieses Metamodells ein systematisches Vorgehensmodell vom Architekturentwurf bis hin zur Codegenerierung und Integration. Darüber hinaus kann unter Verwendung des AUTOSAR UML Profils auch der Einsatz gängiger Modellierungswerkzeuge zur Spezifikation von AUTOSAR-Systemen herangezogen werden.

Weiterhin strebt die AUTOSAR-Initiative nicht nur eine Standardisierung der Schnittstellen der verschiedenen Komponenten untereinander, sondern auch zu den darunter liegenden Service-Routinen der Laufzeitumgebung an, wie z.B. Buskommunikation, Betriebssystemdienste und Hardwaretreiber. Die hierfür spezifizierte Schichtenarchitektur stellt so eine einheitliche Ablaufplattform und transparente Kommunikationsschnittstelle für alle AUTOSAR-Komponenten im System dar.

## 2.1 Standardisierungen

Die Einführung einer allgemeinen Software-Basis für vernetzte Steuergeräte in Automobilen schafft ähnliche Grundlagen, wie sie bei Betriebssystemen im Bereich der Personal Computer schon seit längerem gegeben sind. Die Motivation für eine Standardisierung beruht auf der Anforderung, komplexere (elektronische) Systeme in Fahrzeugen entwickeln zu können und die erarbeiteten Lösungen über Produktgrenzen hinaus mit geringerem Aufwand als bisher anpassen

und verwerten zu können. Qualität und Betriebssicherheit der Fahrzeugelektronik können verbessert werden.

Standardisierungen wurden in verschiedenen Bereichen der Entwicklung eingeführt. Einbezogen wurden beispielsweise die Art der Designspezifikation sowie eine Basis-Software für Steuergeräte und Schnittstellen. Die Standardisierung der Formate zum Austausch von Spezifikationen schafft die Voraussetzungen für die Rückverfolgbarkeit (Traceability) von funktionalen Anforderungen. Dies kann auch der Verbesserung der Spezifikationen in Format und Inhalt dienen. Darüber hinaus sollte auch die Anzahl an kompatiblen Tools erhöht werden, so dass eine durchgängige Tool-Kette die Entwicklung unterstützen kann.

Eine Basis-Software vermeidet den Arbeitsaufwand für Implementierungen und Optimierungen von Komponenten, die für den Kunden keinen erkennbaren Wert haben. Die Arbeit kann auf die wettbewerbstauglichen Funktionen konzentriert werden. Eine ausgereifte Basis-Software erhöht die Qualität der Software insgesamt.

Auch die Standardisierung der Schnittstellen sorgt dafür, dass Anpassungen von Funktionen an Hersteller-spezifische Betriebsbedingungen entfallen, die ebenfalls keine Vorteile im Wettbewerb darstellen. Ohne standardisierte Schnittstellen können kleinste Verbesserungen an einzelnen Komponenten einen unverhältnismäßig hohen Aufwand verursachen, da häufig die Schnittstellen anderer Komponenten nachgearbeitet werden müssen. Die Hersteller-übergreifende Wiederverwendbarkeit von Modulen wird durch die Standardisierung der Schnittstellen ebenso begünstigt, wie die Austauschbarkeit von Komponenten, die von verschiedenen Zulieferern stammen. Das Implementieren von Software-Funktionalitäten, die nicht an eine bestimmte Hardware gebunden sind, wird durch generische Schnittstellenkataloge erleichtert.

Tools zur automatisierten Erzeugung von AUTOSAR-konformem Code sind auf standardisierte Schnittstellen angewiesen. Mit ihnen wird auch die modellbasierte Entwicklung vereinfacht. Klare Schnittstellen zwischen der Basis-Software und dem Code, der aus Modellen erzeugt wird, sind zwingend erforderlich. Durch die Standardisierung wird ein starkes Anwachsen der Zahl von Schnittstellen, die zwischen Automobilherstellern und Zulieferern definiert werden müssen, vermieden.

## 2.2 Hardwareunabhängige Software-Entwicklung

AUTOSAR sieht einen möglichst hohen Abstraktionsgrad von der Hardware und damit auch von Prozessoren und Mikrocontrollern vor. Nicht mehr verfügbare Mikrocontroller-Modelle verursachen einen hohen Aufwand bei der Anpassung der bestehenden Software an alternative Modelle. Die Erweiterung des Funktionsumfanges der Software erfordert häufig eine Erhöhung der Prozessorleistung

und kann zu einer unausweichlichen Überarbeitung des Hardware-Designs führen. Mit der Entwicklung nach dem AUTOSAR-Standard sollen beim Austausch des Mikrocontrollers keine Anpassungen auf höheren Software-Ebenen vorgenommen werden müssen.

Diese Konzeption beinhaltet eine Laufzeitumgebung mit der Bezeichnung RTE[1] (s. Abschnitt 2.5), die durch die Kapselung ihrer Funktionen für eine Unabhängigkeit von der Kommunikationstechnologie sorgt. Auch hier erleichtern Standardisierungsmaßnahmen die Kommunikation. Die RTE soll den Aufwand bei der Übertragung von Funktionen auf andere Steuergeräte vermindern. Neben der begünstigten Wiederverwendbarkeit soll die RTE auch für eine einfache Aufteilbarkeit und Verschiebbarkeit von Funktionen sorgen.

## 2.3 Abstraktionsschichten von AUTOSAR

Zur Umsetzung von Systemfunktionalitäten unter Berücksichtigung bestimmter Qualitätsanforderungen schlägt der AUTOSAR-Standard eine ganzheitliche Vorgehensweise für die modellgestützte Entwicklung einer geeigneten Systemarchitektur bis hin zur tatsächlichen Implementierung vor [AUT06]. Zu diesem Zweck definiert der AUTOSAR-Standard neben einem Metamodell zur Beschreibung kompletter Systemarchitekturen ein Architektur-Schichtenmodell mit standardisierten Schnittstellen, das die verschiedenen Abstraktionsebenen der Architektur integriert und zunächst von der Hardware-Plattform und Kommunikation abstrahiert. Abbildung 2.1 zeigt die Abstraktionsschichten aus Sicht der Software-Architektur und nach dem Mapping der Softwarekomponenten auf die Steuergeräte. Das Verhalten einer Softwarekomponente wird von darunter liegenden Schichten bezüglich der Kommunikation durch den Virtual Function Bus (VFB) und dem Hardware-Zugriff über Basic Software (Systemdienste) abstrahiert.

Der VFB bietet also neben der Umsetzung eines systemweiten Kommunikationsprotokolls eine standardisierte Schnittstelle für den Zugriff auf die Funktionen der Basic Software (Betriebssystem-Dienste wie z. B. Treiber) und ermöglicht so die Integration aller Softwarekomponenten im System. Somit bildet der VFB eine einheitliche, transparente Kommunikationsschicht zwischen Softwarekomponenten, unabhängig vom späteren Ausführungsort dieser Komponenten.

Nach der Implementierung des Systems, also der Verteilung und Ausführung der Komponenten und deren Kommunikation auf verschiedenen Steuergeräten des Bordnetzes wird für jedes Steuergerät spezifisch eine Implementierung des VFB als RTE (Runtime Environment) notwendig, der die tatsächlich von den Komponenten benötigten Dienste und Kommunikationsverbindungen umsetzt.

[1] Runtime Environment

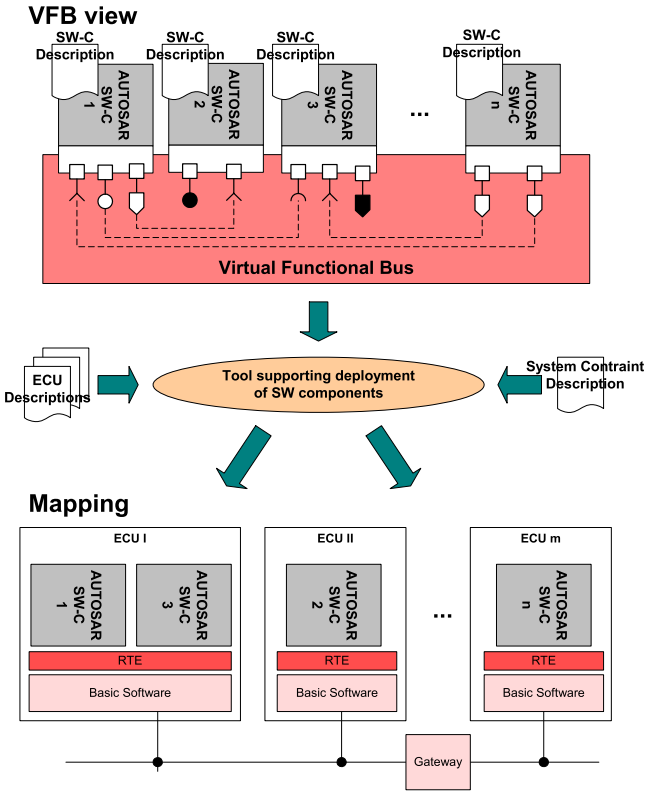


**Abbildung 2.1:** Abstraktionsschichten von AUTOSAR

Jede ECU erhält somit je nach Anforderungen der darauf laufenden Softwarekomponenten eine spezifische und optimierte Laufzeitumgebung.

Als Abstraktionsmodelle zur Beschreibung und Integration der verschiedenen Sichten auf die Systemarchitektur gemäß den zugehörigen Ausschnitten aus dem Metamodell definiert der AUTOSAR-Standard fünf Diagrammarten. Darauf aufbauend wird eine einheitliche Vorgehensweise (AUTOSAR-Methodik) vorgeschlagen. Die wesentlichen Schritte dieser Methodik und die zugehörigen Diagramme, die für die Aspekte der hier zu betrachtenden System- und Softwarearchitektur von Bedeutung sind, werden später beschrieben.

## 2.4 Die AUTOSAR-Methodology

Die AUTOSAR-Methodology (AUTOSAR-Methodik) [AUT06] beschreibt den Ablauf des automotiven Entwicklungsprozess in mehreren Phasen. Hierzu kommen in den unterschiedlichen Phasen oftmals verschiedene Werkzeuge zum Ein-

satz, deren Austauschbarkeit durch festgelegte Formate ermöglicht wird. Die Abbildung 2.2 zeigt die Phasen und Abläufe der Methodik.

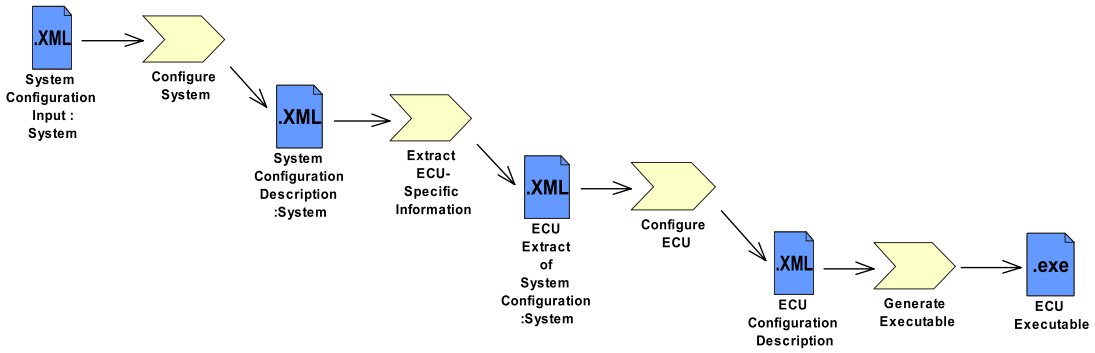


**Abbildung 2.2:** AUTOSAR-Methodology [AUT07a]

Die AUTOSAR-Methodology legt fest, zu welchen Zeitpunkten und zu welchen Zwecken Beschreibungen im Entwicklungsprozess verwendet werden. Insbesondere werden die Arbeitsschritte betrachtet, die für den Modellaustausch in der Software-Entwicklung notwendig sind. Die Methodik beschreibt dabei nicht die Rollen und Zuständigkeiten *innerhalb* des Entwicklungsprozesses.

Die Methodik definiert zwei große Bereiche zum Entwurf eines Systems. Im ersten Teil wird der Entwurf der gesamten Software des Fahrzeugs beschrieben, dies geschieht auf Systemebene. Anschließend werden die Arbeitsschritte auf der Ebene eines einzelnen Steuergeräts erläutert [AUT09a]. Diese Zweiteilung wird in der Grafik 2.3 durch eine Trennlinie gekennzeichnet.

### Systemkonfiguration

Zunächst wird die Systemkonfigurations-Eingabeinformation definiert. Hierbei werden die Software- und die Hardware-Komponenten ausgewählt und die systemweiten Beschränkungen identifiziert. Aus praktischer Sicht bedeutet die Definition der Systemkonfigurations-Eingabeinformation, das Ausfüllen oder Bearbeiten von entsprechenden Vorlagen (AUTOSAR-Templates).

Die System-Konfiguration beschreibt das Zuordnen (Mapping) von Software-Komponenten auf Steuergeräte. Das Mapping ist eine der wichtigsten Entscheidungen im Entwicklungsprozess, die die Erfüllung der Vorgaben zum Ressourcen-Verbrauch und Echtzeit-Bedingungen stark beeinflusst. Die Konfiguration erfolgt in drei Schritten:

- Beschreibung der System-Konfiguration mit allen Informationen (Mapping, Hardware-Topologie)

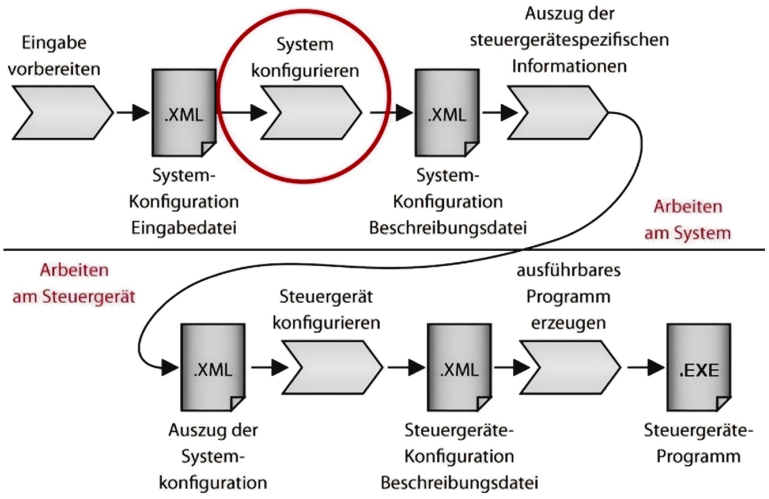


**Abbildung 2.3:** Zweiteilung der AUTOSAR-Methodology [AUT07a]

- Zuordnung jeder Software-Komponente zu einem Steuergerät
- Beschreibung der Eigenschaften der verwendeten Netzwerke und Medien über die Kommunikationsmatrix

Durch das Extrahieren steuergerätespezifischer Informationen, werden die Daten für die Ausführung der Software eines bestimmten Steuergeräts aus der Beschreibung der System-Konfiguration ermittelt (letzte Schritt vor der Trennlinie in Abb. 2.3). Anschließend ist mit der Konfiguration des Steuergeräts fortzufahren [AUT09a].

### Steuergerätekonfiguration

In der Steuergerätekonfiguration werden alle notwendigen Informationen für die Implementierung (z. B. Task Scheduling, benötigte Basis-Software-Module und deren Konfiguration) hinzugefügt. Dieser Schritt liefert die Beschreibung der Steuergerätekonfiguration, welche wiederum für den letzten Schritt der AUTOSAR-Methodology benötigt wird. Hier wird der Programm-Code des jeweiligen Steuergeräts aus der gegebenen Beschreibung generiert [AUT09a].

Bei jeder Änderung oder Ergänzung von Software-Komponenten auf einem Steuergerät oder der Modifikation des Kommunikationsnetzwerks (z. B. durch Beschränkungen) zwischen den Steuergeräten, wird die Steuergerätekonfiguration angepasst [AUT09a].

### 2.4.1 Konzepte des Architekturentwurfs

Der Architekturentwurf stellt nicht nur, wie vorhergehend erwähnt, in der AUTOSAR-Methodology einen zentralen Schritt dar, sondern ist auch in den hier später vorgestellten Konzepten zur Qualitätssicherung von besonderer Bedeutung. Darum soll im Folgenden zunächst allgemein auf den Begriff Architektur und später auf den Architektur-Entwurf im Zusammenhang mit AUTOSAR eingegangen werden.

Der Begriff *Architektur* umfasst Abstraktionssichten auf ein System, die die grundlegenden strukturellen Eigenschaften dieses Systems beschreiben [BCK03]. Dabei wird ein System als Menge von Hardware- und Software-Elementen (Komponenten) und deren Beziehungen untereinander (z.B. durch Kommunikation) beschrieben. Von der tatsächlichen internen Realisierung (z.B. Implementierung von Softwarekomponenten oder eines Kommunikationsprotokolls) wird dabei abstrahiert, so dass nur Verhalten und Eigenschaften dieser Elemente, die nach außen (z.B. für andere Elemente) sichtbar bzw. zugreifbar sind, in Form von Schnittstellenbeschreibungen spezifiziert werden.

Wird die geforderte Gesamtfunktionalität eines konkreten Fahrzeugs betrachtet, so kann eine Vielzahl von Varianten zur Umsetzung individueller Anforderungen herangezogen werden. Dieser Architekturentwurf, der somit die früheste Designentscheidung im Entwicklungsprozess darstellt, bildet demnach die Grundlage für alle nachfolgenden Entwurfsschritte bis hin zur Implementierung. Gemäß dieser Zielsetzung verfolgen Ansätze zur Spezifikation von Systemarchitekturen zwei Grundsätze:

1. Die Konzepte sollen allgemein sein und von Implementierungsdetails abstrahieren.

2. Die Konzepte sollen einfach und grundlegend sein, aber zugleich wesentliche Grundstrukturen des Gesamtsystems festlegen.

Hierzu zählen in der Regel Konstrukte zur Kapselung, Dekomposition und Hierarchisierung von Systemartefakten grober Granularität. Darauf aufbauend können entsprechende Architektur-Konzepte des modellbasierten Software-Engineering zum Einsatz kommen, wie beispielsweise komponentenbasierte Ansätze. Eine Komponente bildet hier das zentrale Abstraktionskonzept zur Spezifikation der grundlegenden Systembestandteile, die ohne weitere Anpassungen an vorgesehenen Stellen eingesetzt, mit anderen Komponenten kombiniert und zu einem Gesamtsystem integriert werden können. Dies erfolgt über die Definition invarianter, bekannter Schnittstellen zur Festlegung der durch die Komponente gelieferten Teilfunktionalität im System, die somit die Verwendung der Komponente beschreibt, ohne jedoch deren interne Implementierung vorzugeben. Insbesondere kann es sich je nach Sicht auf die Architektur sowohl um Hardware-Komponenten,

| **Einflussquelle** | **Faktoren, Einflüsse** |
|---|---|
| Stakeholder | Use Cases, Requirements, Constraints |
| Auftraggeber | Kosten, Entwicklungsziele, Produktlinien |
| Architekten | Erfahrungen, Patterns |
| Technisches Umfeld | Ausstattung, Vorgaben, Standards |

**Tabelle 2.1:** Architekturentwurf – Einflüsse

wie beispielsweise Steuergeräte und Busse, als auch Bestandteile der Software des Systems handeln. Im zweiten Fall beschreiben die Komponenten *„Building Blocks“* der Gesamtsoftware, die auf dem Steuergerätenetz verteilt sind, und bestimmte (logische) Teilaufgaben erfüllen.

Diese Sichtweise auf ein zu entwerfendes System bietet mehrere Vorteile. Durch die Spezifikation der Systemarchitektur werden bereits in einer frühen Phase der Entwicklung grundlegende Festlegungen bezüglich der strukturellen Eigenschaften gemacht. Hierdurch können in der Regel zwar nicht endgültige Aussaugen über die Qualität des fertigen Systems gemacht werden, aber durch falsche Architekturentscheidungen kann durchaus die Erfüllung solcher Kriterien von vornherein ausgeschlossen sein. Somit werden Qualitätsattribute für ein System maßgeblich von der Architektur mitbestimmt.

Entsprechend stellt die Architektur eines Systems die früheste Design-Entscheidung im Entwicklungsprozess dar und hat somit weit reichende und schwer zu korrigierende Auswirkungen auf die folgenden Entwicklungsphasen.

Weiterhin erlaubt die Komponenten-orientierte Architekturspezifikation einen strukturierten inkrementellen Entwicklungsprozess in den folgenden Phasen (Design, Implementierung) sowie den Einsatz von Techniken wie z. B. Rapid Prototyping, in dem beispielsweise Komponenten evolutionär durch konkrete Implementierung ersetzt werden.

Wie in Tabelle 2.1 dargestellt, beeinflussen eine Vielzahl von Faktoren und an der Entwicklung eines Systems Beteiligte (Stakeholder) die Architektur für ein zu entwickelndes System. Zur angemessenen Berücksichtigung dieser unter Umständen gegenläufigen Einflüsse müssen Architekturvarianten spezifiziert werden, die einen geeigneten Tradeoff (Kompromiss) zwischen diesen Faktoren gewährleisten.

Folglich gibt es keine eindeutige optimale Architektur zur Umsetzung einer geforderten Systemfunktionalität, sondern diese hängt von der Auswahl und Gewichtung von bestimmten Qualitätsattributen (Qualitätskriterien) ab, die vom Gesamtsystem gefordert werden. Weiterhin ist die Bewertung konkreter Qualitätsattribute für eine bestimmte Systemarchitektur in der Regel abhängig vom Anwendungskontext, den Anwenderprofilen, Anwendungsszenarien etc., also von Faktoren, die je nach Einsatz und Umfeld des Systems variieren können.

Nachdem wir die grundlegenden Konzepte des Architekturentwurfs beschrieben haben, soll nun auf die Besonderheiten von Architekturen automotiver Steuergerätenetze eingegangen werden.

Stetig steigende Anforderungen an (Software-) Systeme führen zu immer komplexeren Architekturen für die Umsetzung solcher Systeme. Das gilt im besonderen Maße auch für die Entwicklung von Steuerungssystemen im Automobilbereich [SZ06]. Die hier betrachteten, stark verteilten, eingebetteten Systeme weisen eine Vielzahl spezialisierter Software- und Hardware- Einheiten auf, für die sich eine entsprechend hohe Zahl komplexer Kombinationsmöglichkeiten ergeben. Gleichzeitig sollen die Kosten so gering wie möglich gehalten werden, da hohe Stückzahlen produziert werden müssen.

Gerade im Bereich der automotiven Systeme mit ihren komplexen Elementen zur Steuerung und Bordnetzen mit einer Vielzahl heterogener Bauteile und hohen Qualitätsanforderungen, dient die abstrahierende Architektursicht auf die Systemstruktur als übersichtliche Kommunikationsgrundlage für alle an der Entwicklung beteiligten Parteien.

Eine Folge der wachsenden Anforderungen an die Funktionalität im Automobil ist der Übergang von einer ehemals Steuergeräte-orientierten hin zu einer verstärkt Funktions- und somit Software-orientierten Entwicklung des Steuerungssystems unter besonderer Berücksichtigung der Kommunikationsaspekte.

Dabei verfolgen Architekturmodelle zur Spezifikation der grundlegenden Systemstrukturen grundlegende Strategien (Taktiken), um die Komplexität derartiger Systeme insbesondere in frühen Phasen der Entwicklung beherrschen zu können:

1. **Dekomposition und Hierarchisierung:** Komponenten-orientierte Beschreibung der funktionalen Architektur. Grundlegende Systemfunktionen werden in unabhängigen Teilsystemen (Komponenten oder Funktionsblöcken) gekapselt. Kommunikation und Interaktion zwischen Komponenten findet ausschließlich über invariante Schnittstellen in festgelegter Richtung statt, so dass sie heterogen und parallel entwickelt und später beliebig modifiziert und ausgetauscht werden können. Darüber hinaus können allgemein definierbare Komponenten- Typen mehrfach im System eingesetzt und auf spezielle Anforderungen für eine bestimmte Rolle im System durch Variantenbildung angepasst werden. Die Bildung von Komponentenhierarchien durch Komposition (logische Gruppierungen) der funktionalen Struktur des Gesamtsystems trägt ebenfalls zur besseren Beherrschbarkeit der Komplexität bei.

2. **Abstraktion:** Beschreibung von Systemeigenschaften durch abstrahierende Modelle. Die Sicht auf die Architektur des Systems beschreibt die Strukturierung grundlegender Systemelemente (Software, Hardware), ihrer Interaktion und dafür notwendigen Verbindungen untereinander sowie ihre

Verteilung im System. Dabei wird von Implementierungs- und technischen Details der Elemente und Kommunikation abstrahiert.

Dieser Ansatz ermöglicht es dem Systemarchitekten, frühzeitig strukturelle Entscheidungen für das System unabhängig von der Implementierung der Funktionen und der technischen Realisierung der Kommunikation zu treffen.

Auf Grundlage solcher Komponenten-basierter Architekturspezifikationen werden dann im Allgemeinen folgenden Design-Prinzipien verfolgt:

- Trennung von Daten produzierenden Modulen (z.B. Sensoren) und Daten konsumierenden Modulen (z.B. Aktuatoren). Eine Ausnahme bilden dabei beispielsweise Kontrollmodule auf der obersten Ebene einer hierarchischen Komponente, die Daten von außen entgegen nehmen und an die entsprechenden Untermodule zur Weiterverarbeitung delegieren.

- Prozesse sind modulübergreifend. Die Implementierung der einzelnen Funktionalitäten in späteren Phasen der Entwicklung auf Grundlage der Architektur erfolgt durch Kombination (Interaktion, Kommunikation) von Funktionen mehrerer Komponenten. Beispielsweise wird eine Regelungsaufgabe zumindest ein Sensormodul zur Ermittlung des Ist-Wertes und ein Aktuatormodul zum Setzen des Stellwertes benötigen.

- Die Interaktion zwischen Modulen wird durch eine kleine Zahl einheitlich durchgängiger Kontrollmuster beschrieben. Beispiele hierfür sind die Verwendung von Message-Passing für Daten-orientierte Kommunikation und der Client-Server-Ansatz für Operations-orientierte Kommunikation. Diese beiden Paradigmen sind auch im AUTOSAR- Standard umgesetzt.

Zur Beschreibung der grundlegenden Prinzipien und Muster bei der Spezifikation einer Systemarchitektur können folgende Terminologien unterschieden werden:

1. Architektur-Pattern: Muster dienen der Einschränkung der Menge potenzieller Architekturvarianten für ein System, so dass nur bestimmte Elementtypen, deren Nutzung und Beziehung untereinander zugelassen werden. Ein Beispiel hierfür ist das Client- Server-Pattern, das zwei bestimmte Arten von Elementen zulässt, deren Verhalten bezüglich ihrer Interaktion untereinander festgelegt ist.

2. Referenz-Modell: Beschreibung von Referenz-Strukturen für Architekturen, die für wiederkehrende Standardfunktionen eines Systems verwendet werden sollten. Das Modell legt die Dekomposition des Systems und den Datenfluss zwischen den sich dadurch ergebenden Teilen fest. Die Kooperation dieser Teile löst das entsprechende Gesamtproblem.

In der Regel werden Systemarchitekturen nicht durch ein Gesamtmodell beschrieben, sondern es werden unterschiedliche Sichten (Views) auf die Systemstruktur definiert, die jeweils einen bestimmten Aspekt des Systems betrachten und dabei von anderen Details abstrahieren. Die Systemarchitektur ergibt sich schließlich aus der Integration dieser Teilsichten.

Die konkrete Implementierung der Softwarefunktionen innerhalb der Komponenten kann dann beispielsweise in Form von Funktionsblöcken in MATLAB/Simulink/Stateflow und anschließender Codegenerierung erfolgen. Gleiches gilt für die konkrete Realisierung der in der Hardwarearchitektur beschriebenen technischen Ausführungsplattform: Von Details zu den Einbauorten, der Verkabelung, Stromversorgung etc. wird in der Phase abstrahiert.

Dieses Sichtenkonzept, zusammen mit dem komponentenbasierten Entwicklungsansatz (Component Based Software Engineering, CBSE), erlaubt es dem Systemarchitekten nicht nur die Komplexität des Gesamtsystems beherrschbar zu halten. Darüber hinaus ermöglicht dieser Ansatz einen nahtlosen Austausch (Varianten) sowie die Wiederverwendung (Reuse) einzelner Komponenten in Systemen, sowohl aufseiten der Software als auch der Hardware.

Nachfolgend soll auf das Konzept der Abstraktion durch Architekturlevel näher eingegangen werden. Je nach Umfeld und Quellenauswahl existiert eine Vielzahl unterschiedlicher Abstraktionssichten und damit verbundener Terminologien zur Beschreibung von Systemarchitekturen. Im Bereich der automotiven Systeme beschränken sich die Abweichungen zwischen verschiedenen Ansätzen aber in der Regel auf Details, so dass folgende Abstraktion für die Beschreibung einer Systemarchitektur als wesentlich angesehen werden kann:

> Strukturierte Darstellung des Steuergerätenetzwerks und der Verteilung der zu realisierenden (Software-) Funktionen darauf

wobei von der konkreten Implementierung dieser Funktionen und deren Kommunikation untereinander abstrahiert wird. Die Menge dieser Funktionen setzt dann die geforderte Gesamtfunktionalität des Systems um. Diese bildet dabei die Grundlage für verschiedene Architekturvarianten und deren Vergleich durch Evaluation und wird im Folgenden als invarianter Ausgangspunkt für alle weiteren Betrachtungen angenommen. Sie steht am Beginn des Entwurfsprozesses und beschreibt aus Benutzersicht Ausstattungsmerkmale (Features) des Automobils und deren für den Benutzer sichtbaren Verhalten und Eigenschaften. Ausgehend von der oben genannten Definition lassen sich im Wesentlichen die folgenden Abstraktionssichten und darin betrachteten Elemente identifizieren:

1. **Funktionale Architektur:** Beschreibung der (logischen) Software-Architektur für die Umsetzung der Systemfunktionalitäten. Durch funktionale Dekomposition werden hierarchische, miteinander interagierende Software-Komponenten (Funktionsblöcke) definiert, die jeweils Teilfunktionen für

die Umsetzung der Gesamtfunktionalität beisteuern. Interaktion zwischen nach außen abgeschlossenen Komponenten ist nur über feste Punkte (Ports) möglich, für die eindeutige Schnittstellen festgelegt werden. Diese beschreiben Datenpfade (Konnektoren) zwischen Funktionen zweier Komponenten und zugehörige (logische) Signale oder Datenwege anhand der Kommunikationsrichtung und der Typen dieser Daten.

Von der Implementierung der Funktionen und ihrer Schnittstellen wird ebenso abstrahiert wie von der (technischen) Umsetzung der Kommunikation zwischen den Funktionsblöcken. Die Beschreibung der internen Abläufe der Komponenten kann später anhand von Modellen (z.B. MATLAB/Simulink) erfolgen und/oder direkt in Code für die entsprechende Zielplattform umgesetzt werden.

2. **Hardware-Architektur:** Zu den Festlegungen für das Steuergerätenetzwerk gehört die Auswahl, Art und Anzahl von Steuergeräten (ECUs) im System, deren Ausstattung, wie z. B. Prozessoren, Speicher, Sensoren, Aktuatoren sowie deren topologische Anordnung im System. Für die technische Realisierung der Kommunikationen zwischen den Hardware- Elementen im System, beispielsweise unter Verwendung eines Bussystems (CAN, LIN, FlexRay, etc.), müssen entsprechende Elemente (Leitungen, Hubs, etc.) angeordnet werden. Schließlich werden in dieser Architektur-Sicht auch die Einbauorte und die Verschaltungen der Hardware-Einheiten spezifiziert.

3. **System-Architektur:** Die Gesamtarchitektur des Systems ergibt sich durch die Integration der funktionalen Architektur und der Hardware-Architektur. Hierfür wird die Verteilung (Mapping) der (Software-) Funktionen (Komponenten) auf die für ihre Ausführung vorgesehenen Steuergeräte vorgenommen. Die sich hieraus ergebenden Kommunikationswege (Kommunikationsmatrix), d.h. das Mapping der Datensignale zwischen Funktionen, wird durch entsprechende Signale (Frames) des gewählten Bus-Protokolls umgesetzt.

### 2.4.2 Architekturentwurf im AUTOSAR-Kontext

Die AUTOSAR-Initiative strebt nicht nur eine Standardisierung der Schnittstellen der Komponenten untereinander, sondern auch zu den darunter liegenden Service-Routinen der Laufzeitumgebung an, wie z.B. Buskommunikation, Betriebssystemdienste und Hardwaretreiber. Die hierfür spezifizierte Schichtenarchitektur stellt so eine einheitliche Ablaufplattform und transparente Kommunikationsschnittstelle für alle AUTOSAR-Komponenten im System dar. Durch

die Software-orientierte und modellbasierte Entwicklung können bewährte Techniken des Software-Engineering bei der Entwicklung von AUTOSAR-Systemen angewendet werden, wie z.B.:

1. Konzepte der *Model Driven Architecture* (MDA)

2. Plattformunabhängige Konzeption von Komponenten-Interaktion durch einen Middleware-Ansatz

3. Werkzeuggestützte Entwurfsmethodik zur flexiblen und strukturierten Systementwicklung:

   - Plattformunabhängiges Systemmodell zur Applikationsmodellierung (Software Components und Virtual Function Bus)
   - Plattform Deployment (Codegenerierung für die Softwarefunktionen einzelner Steuergeräte sowie eigene Runtime Environment und Basic Software)

Das AUTOSAR-Metamodell ermöglicht die Beschreibung von Software, Hardware und deren Zusammenhänge in einem Automobil mithilfe von unterschiedlichen Architekurleveln. Zudem beschreibt die AUTOSAR-Methodik einen Top-Down-Ansatz zur systematischen Spezifikation der Systemstrukturen gemäß dem Abstraktionsgrad des entsprechenden Levels und schließlich deren Integration in eine Gesamtarchitektur. Dabei stellt die Betrachtung der Systemarchitektur, also die implementierungsunabhängigen Eigenschaften eines zu entwickelnden Systems, nur einen Teil der AUTOSAR-Methodik dar.

Mit dem UML Profile von AUTOSAR [AUT07a] ist es möglich, AUTOSAR-konforme Architekturen mithilfe der bekannten Konstrukte von der Unified Modeling Language zu spezifizieren. Zudem gibt es eigene graphische Notationsmittel für die unterschiedlichen Architekturelemente.

Ausgehend von einer auf diesem Wege definierten Architektur-Variante zur Umsetzung einer (invarianten) Gesamtfunktionalität können dann Evaluationsverfahren zur Bewertung des Entwurfs angewendet werden.

Ausgangspunkt für die Spezifikation einer Systemarchitektur ist eine (in der Regel halbformale) Beschreibung der geforderten Funktionalität des Systems (Liste von Features/Ausstattungsmerkmalen) sowie eventuell weitere zu beachtende Eigenschaften (z.B. Zeitverhalten oder Sicherheitsaspekte) und Einschränkungen (Constraints) bei deren Umsetzung, z.B. eingeschränkte Auswahl von Hardware-Ressourcen. Die Spezifikation einer Systemarchitektur durch den AUTOSAR-Entwicklungsprozess zur Umsetzung dieser Funktionalität sieht die nachfolgend beschriebenen Schritte vor.

### Funktionale Softwarekomponenten-Architektur

Eine (Software-) Komponente ist eine vollständige, wiederverwendbare, unabhängige Softwareeinheit mit eindeutig definierten Schnittstellen zur Integration in eine Umgebung bestehend z.B. aus anderen Komponenten. Im Rahmen der funktionalen Architektur ist innerhalb einer Komponente ein Teil der Funktionalität des Systems umgesetzt, wobei von den inneren Details (Implementierung der Schnittstellen) abstrahiert wird. Durch die Abstraktionen des AUTOSAR-Ansatzes in der hier betrachteten Schicht kann später die interne Realisierung also sowohl modellbasiert beschrieben als auch direkt in Form von Code vorgenommen werden.

Die Gesamtfunktionalität wird durch Komposition und Hierarchisierung von Komponenten realisiert. Diese erlauben eine logische Strukturierung und Kapselung von (Software-) Funktionen unabhängig vom Steuergerätenetz (Steuergeräte, Leitungen, Busstandards, Speicherzugriff, etc.), auf dem sie später ausgeführt werden und mit anderen Komponenten interagieren.

#### Software Component Diagrams (SW-C):

Der AUTOSAR-Standard sieht Software Component Diagrams (SW-C) zur Beschreibung der Softwarekomponenten-Architektur vor.

Softwarekomponenten sind die Hauptbestandteile dieser Diagramme zur Beschreibung der funktionalen Architektur unter Annahme eines transparenten Kommunikationsflusses (Kontroll- und Datenfluss) zwischen diesen Komponenten. Grundsätzlich werden zwei Arten von (Software-) Komponenten unterschieden: Atomare Komponenten und Kompositionen.

**Atomare Komponenten:** Diese Komponenten kapseln eine nicht weiter unterteilbare Grundfunktionalität des Systems. Sie stellen einen Teil des Softwaresystems dar, der später tatsächlich in einer Programmiersprache als Funktionseinheit implementiert wird und deshalb zur Ausführung nicht auf mehrere ECUs verteilt werden kann. Spezialformen atomarer Komponenten sind Komponenten, die das auf der Software-Ebene sichtbare Verhalten von Sensoren (Datenquellen) und Aktuatoren (Datensenken) repräsentieren. Im Allgemeinen wird eine beliebige Komponente im System durch ein Rechteck dargestellt und die Art der Komponente wird durch ein graphisches Symbol spezifiziert. Die Beschriftung definiert den Namen und Typ einer Komponente.

Für den Fall, dass die Angabe eines Namens für den Prototypen entfällt, handelt es sich um die Darstellung eines Komponenten-Typen (siehe nachfolgend). Sensor- bzw. Aktuatorkomponente sind spezielle atomare Komponenten, und werden in der Regel durch eine Pfeilnotation in der rechten oberen Ecke spezifiziert.

Im Allgemeinen können beliebige Software-Bestandteile des Systems als Komponenten angesehen werden, wie beispielsweise auch Schnittstellen zu Hardware-Treibern. Da der hier betrachtete Architektur-Level von der unterliegenden Hardware abstrahiert, werden im Folgenden nur AUTOSAR-konforme Softwarekomponenten betrachtet.

**Kompositionen:** Kompositionen sind logische Strukturierungsmittel, die beliebig viele (Sub-) Komponenten enthalten können und deren Verbindungen untereinander und aus der Komposition herausführend beschreiben. Somit stellt eine Komposition eine komplexe Applikation dar, dessen Funktionalität sich aus der Aggregation der in ihr enthaltenen Komponenten und zugehörigen Teil- Funktionen ergibt. Die Subkomponenten können atomare Komponenten und weitere Kompositionen sein, so dass beliebige hierarchische Gruppierungen beschrieben werden können. Kompositionen besitzen keine Entsprechung mit Blick auf die Implementierung des Software-Systems und können somit gemäß des Mappings ihrer Bestandteile auf mehrere ECUs verteilt werden.

Die grafische Notation für eine zusammengesetzte Komponente besteht in der Regel aus drei Rechtecken in der oberen rechten Ecke der Komponente. Kompositionen können zur Definition beliebiger Hierarchien durch Schachteln von Unterkomponenten verwendet werden. Der Zusammenhang zwischen Kompositionstyp und -prototyp (main) ist hier analog zu den vorherigen Beschreibungen.

Eine Spezialform einer Komposition stellt die Top-Level-Komponente eines Software- Systems dar, die keine weiteren Verbindungen zu anderen Komponenten aufweist, und das gesamte Softwaresystem des Automobils enthält. Wie bereits oben erwähnt, werden Verbindungen zu speziellen Komponenten, wie z.B. Treibern, an dieser Stelle (noch) nicht betrachtet.

Beispiele für die verschiedenen Komponenten-Arten sind in Abschnitt über die Beschreibung der Fallstudie dargestellt. Neben den Komponenten und deren Strukturierung spezifiziert das Software-Component-Diagram auch deren Verbindungen untereinander, was über Ports, Konnektoren und Schnittstellen beschrieben wird.

**Ports und Konnektoren:** Ports einer Komponente stellen Verbindungspunkte zu dieser Komponente für andere Komponenten dar. Eine (gerichtete) Verbindung zwischen kompatiblen Ports zweier Komponenten wird durch einen Konnektor hergestellt. Damit können explizite Abhängigkeiten gemäß der Richtung des Konnektors als *utilize*-Beziehung zwischen diesen Komponenten definiert werden. Für eine Komponenten-orientierte Systemspezifikation ist entscheidend, dass diese Konnektoren die einzigen Abhängigkeiten zwischen ansonsten unabhängigen Komponenten darstellen. Welche Art von Interaktion zwischen verbundenen Komponenten besteht, wird lediglich durch die Schnittstellen-Beschreibung (Interface) eines Ports festgelegt. Im Rahmen des AUTOSAR-Ansatzes deuten Konnektoren Datenflüsse zwischen den Komponenten der zugehörigen Ports an,

und der Typ eines Ports (siehe unten) legt implizit den zum Datenaustausch notwendigen Kontrollfluss fest.

**Interfaces:** Die Schnittstellenbeschreibung eines Ports legt das Kommunikations-Pattern und den Datenfluss für die zugehörige Verbindung fest, also den Namen und die Signatur der Datenelemente oder Dienste, die zwischen den betreffenden Komponenten ausgetauscht werden. Anhand von zwei Eigenschaften können Interfaces und die darin definierten Portbeschreibungen klassifiziert werden: durch die Richtung und durch das von ihnen unterstützte Kommunikations-Paradigma.

Die erste Eigenschaft beschreibt die Richtungen des Datenflusses bzw. Operationsaufrufes eines Ports bezüglich ihrer Konnektoren: Ein PPort bietet Daten bzw. Dienste an (provide), ein RPort benötigt Daten bzw. Dienste (require) zur Umsetzung seiner internen Funktionen. Ob es sich bei diesen Ports um Daten- oder Dienst-orientierte Verbindungspunkte für Konnektoren handelt, ist durch die Art des Interfaces spezifiziert. Dieses legt das Kommunikations- Paradigma und den damit verbundenen expliziten Daten- und impliziten Kontrollfluss für die Kommunikation fest. Folgende Pattern und zugehörige Interfaces werden vom VFB unterstützt:

1. **ClientServerInterface:** Synchrone operations-orientierte Kommunikation. Ein Client (*RPort*) kann einen Operationsaufruf (Dienst) in einem Server (*PPort*) durch System- und führen und dabei Parameter übergeben. Das Ergebnis der Operation wird vom Server an den Client zurückgegeben. Der Aufruf kann blockierend oder nicht blockierend erfolgen.

2. **SenderReceiverInterface:** Daten-orientierte Kommunikation. Asynchroner Datenfluss vom Daten schreibenden Sender (*PPort*) an beliebige, auf die Daten wartende (lesende) Empfänger (*RPort*).

Die (statischen) Eigenschaften von Schnittstellen eines Ports sind gegeben durch die Menge von Datenelementen, die gesendet und empfangen werden können, bzw. durch die Menge von Operationen eines Servers, die von Clients aufgerufen werden können. Die dynamischen (nach der Instanziierung feststehen) Eigenschaften bestehen aus der Menge instanziierter Konnektoren für diesen Port und die sich daraus ergebenden Kommunikationspartner.

**Definition eines SenderReceiverInterface:** Für Datenelemente, die über ein SenderReceiverInterface gesendet (PPort) oder empfangen (RPort) werden können, werden ein Name und ein Datentyp (siehe unten) spezifiziert. Zusätzlich kann festgelegt werden, ob die Daten gepuffert werden oder eventuell durch Überschreiben verfallen können.

An dieser Stelle soll darauf hingewiesen werden, dass es sich bei diesen Datenelementen gemäß der software-orientierten AUTOSAR-Terminologie nicht um Signale handelt. Vielmehr sieht AUTOSAR eine Vielzahl unterschiedlicher Signal-

Begriffe vor, wie z.B. Software-Signale, die Software-Implementierungen von Kontrollfluss-Informationen darstellen. Des Weiteren gibt es technische Signale, die extern auftretende physikalische Werte repräsentieren und System-Signale, die (physikalische) Träger von Dateneinheiten zum Austausch zwischen verschiedenen ECUs im System beschreiben, und dergleichen mehr.

**Definition eines ClientServerInterface:** Für jede Operation (Dienst), die über ein ClientServerInterface von einem Client (*RPort*) in einem Server (*PPort*) angefordert werden kann, wird jeweils eine Liste zugehöriger Argumente definiert. Zu jedem dieser Argumente gehört ein Datentyp (siehe unten) sowie eine Richtung. Die Richtung kann mit in, out, oder inout angegeben werden und legt fest, ob der Client den Wert des Arguments als Aufruf-Parameter (*in*, *inout*) oder der Server das Argument als Rückgabewert (*inout*, *out*) setzen wird.

Zusätzlich können für Operationen Application-Errors angegeben werden, die bei der Ausführung dieses Dienstes durch den Server auftreten können. Diese sind nicht zu verwechseln mit Infrastructure-Errors, die in den unterliegenden Schichten, z.B. der RTE-API oder Basic Software, auftreten können.

Die konkrete Implementierung der Operationen in Code durch Mapping der Schnittstellen auf eine Programmiersprache ist nicht der Teil der Architekturbetrachtung, da diese von den Eigenheiten einer solchen Zielsprache abstrahiert und deshalb auch möglichst grundlegend gehalten ist. Gleiches gilt unter anderem auch für die zur Verfügung stehenden Datentypen, die im Folgenden kurz beschrieben werden.

**Datentypen:** Zur Typisierung vom gesendeten/empfangenen Dateneinheiten und Argumenten von Operationen stehen die üblichen primitiven Datentypen zur Verfügung: *Char*, *String*, *Integer*, *Real*, *Boolean* und *Opaque* (Bitvektoren). Außerdem können komplexere Datenstrukturen durch *Arrays* (Index-assoziierte Menge von Elementen gleichen Typs) und *Records* (Bezeichner- assoziierte Menge von Elementen beliebigen Typs) konstruiert werden. Zusätzlich können für Datentypen semantische Eigenschaften spezifiziert werden, die beispielsweise Konvertierungsregeln zwischen verschiedenen physikalischen Einheiten vorgeben. Auch die Datentypen sind bei der Betrachtung der (Software-) Architektur unabhängig von der späteren Daten-Implementierung, also deren Umsetzung in einer Programmiersprache und der Darstellung in Hardware-Einheiten, wie z.B. Speicher. Das umfasst unter anderem die Anzahl Bytes zur Darstellung eines Datentyps, das Mapping einer Dateneinheit eines bestimmten Typs in ein CAN-Frame eines Signals und die notwendigen Konvertierungen zur Verarbeitung innerhalb verschiedener ECUs.

**Kompatibilität von Ports:** Nachdem für einzelne (atomare) Komponenten Ports und entsprechende Interfaces definiert wurden, können diese nun zur Spezifikation von Interaktionen mit anderen Komponenten über Konnektoren mit deren Ports verbunden werden. Dabei ist darauf zu achten, dass beide Ports kompatibel sind. Dafür ist zunächst sicherzustellen, dass gemäß der gewünschten Kom-

munikationsrichtung der ausgehende Port ein *RPort* und der andere ein *PPort* ist. Außerdem müssen beide Port-Interfaces das gleiche Kommunikations-Paradigma unterstützen, also entweder beide ein ClientServerInterface oder ein SenderReceiverInterface sein. Dann muss untersucht werden, ob die Interface-Definitionen kompatibel sind, was durch ein entsprechendes Bottom-up-Verfahren überprüft werden kann.

Im Zusammenhang mit Kompositionen sind zwei spezielle Typen von Ports und den damit verbundenen Konnektoren zu unterscheiden:

1. **Assembly-Konnektoren:** Verbindung zwischen einem *RPort* und einem *PPort* zweier in derselben Komposition enthaltenden Komponenten. Diese Konnektoren stellen später Konnektoren im technischen Sinne mit Blick auf das implementierte System dar und werden beispielsweise auf einen Bus abgebildet oder über Shared- Memory-Zugriffe umgesetzt.

2. **Delegation-Konnektoren:** Der Port einer in der Komposition enthaltenden Komponente wird mit einem Port der umgebenden Komposition verbunden, somit nach außen sichtbar und für Komponenten außerhalb der Komposition nutzbar gemacht (Delegation) bzw. von außen kommende Konnektoren an innere Komponenten der Komposition weitergegeben (propagieren). Diese Konnektoren sind rein logische Strukturierungsmittel in der Software-Ebene und haben später keine technische Entsprechung.

Die Verwendung von Delegationen in Kombination mit Assembly-Konnektoren ist in hierarchischen Softwarekomponenten-Architekturen an vielen Stellen von Bedeutung, z. B.:

1. Weitergabe *„globaler“* Sensordaten. Ein Sensor liefert Daten, die für mehrere unterschiedliche Komponenten im System von Bedeutung sind, so dass sich die Sensorkomponente nicht eindeutig einer Softwarekomponente zuordnen lässt, sondern die Daten über entsprechende Konnektoren in alle Komponenten und die darin enthaltenden relevanten Subkomponenten propagiert werden.

2. In einer komplexen Softwarekomponente (Komposition mit mehreren Subkomponenten für verschiedene Teilaufgaben) wird eine zentrale Controller-Komponente eingesetzt, die alle eingehenden Daten von Delegations- Konnektoren entgegennimmt und an die zugehörigen Subkomponenten über Assembly- Konnektoren weiterverteilt. Entsprechend kann auch Datenfluss in die andere Richtung (aus der Komposition heraus) umgesetzt werden.

Im Zusammenhang mit Ports ist folgende Unterscheidungen zu erwähnen: Ein Konnektor verbindet jeweils genau zwei Ports, aber ein Port kann beliebig viele Konnektoren bedienen, wodurch folgende Konstellationen möglich sind:

- Mehrere Clients kommunizieren mit einem Server
- Mehrere Empfänger können auf Daten eines Senders warten

Abschließend sollen noch kurz dynamische Porteigenschaften angedeutet werden. Da diese Kontrollfluss-Aspekte beschreibt, werden sie im Rahmen der Architekturbeschreibung nicht weiter betrachtet. Für Interfaces können eine Menge sog. Modes definiert werden, die Zustände der Komponente bezüglich des Internal-Behavior repräsentiert. Auf zwei Arten kann in der Interface-Definition auf das interne Verhalten in Abhängigkeiten von Modes reagiert werden:

- Der Zustandswechsel eines Modes ist ein Trigger, also ein auslösendes Ereignis für einen internen Ablauf innerhalb der Komponente (siehe *Runnable*).
- Der Zustand eines Modes ist ein Guard, also eine Ausführungsbedingung für einen internen Ablauf in der Komponente (siehe *Runnable*).

Weitere Systemdetails, von denen bei der Betrachtung der Architektur (u.a. durch den VFB) abstrahiert wird, sind z.B. Timing Constraints für die Kommunikation oder die Realisierung der Übergabe von Argumenten. Die eingeführten Konzepte werden später anhand der Fallstudie veranschaulicht, siehe hierzu u.a. Abbildung 4.37.

### Zusammenhang zwischen Typen, Prototypen, Instanzen und Rollen

Bei der Spezifikation der Komponenten-Architektur definiert AUTOSAR ein generisches Typen- Konzept, das zwischen Typen, Prototypen und Rollen (von Instanzen) von Komponenten unterscheidet.

Ein ComponentType definiert allgemeine Eigenschaften einer bestimmten Art von Komponente anhand ihrer Ports und deren zugehörigen Schnittstellen. Diese Typen können als ComponentPrototypes beliebig oft in verschiedenen Kontexten im Gesamtsystem bei der Beschreibung der Software-Architektur verwendet werden.

Alle Komponenten im System mit gleichem Typ stellen prinzipiell gleichartige Komponenten (mit gleichen Schnittstellen) dar. Allerdings können diese Prototypen unterschiedlich instanziiert werden, d. h. ihnen können verschiedene InternalBehavior und ComponentImplementation gemäß dem Metamodell von AUTOSAR zugeordnet werden. Durch diese Beschreibung der internen Abläufe dieser Komponenten, d.h. die Implementierung der Schnittstellen, können verschiedene Rollen für gleiche Prototypen im System festgelegt werden. Neben dem internen Verhalten können sich die Komponenten auch durch abweichende Instanziierungen ihrer Ports unterscheiden, d.h. die Anzahl und Zielkomponenten von Konnektoren eines Ports können beliebig variieren, solange die Schnittstellen

kompatibel sind. Folgendes, gängiges Beispiel aus dem automotive Bereich soll das Prinzip verdeutlichen: Zwei *„symmetrische“* Funktionalitäten (z.B. Steuerungsysteme mit gleichen Funktionalitäten für Fahrer- und Beifahrertür) sollen durch jeweils eine Steuerungskomponente implementiert werden, die beispielsweise auf Sensoreingaben durch entsprechende Regelungsaufgaben auf ihrer jeweiligen Seite reagiert. Da beide grundsätzlich die gleichen Funktionen umsetzen werden, wird ein gemeinsamer Obertyp ControlComponentType mit einheitlicher Schnittstellenbeschreibung (u.a. für die Sensorik) für beide Komponenten definiert. Bei der Beschreibung der Softwarearchitektur wird dann entsprechend für Fahrer- und Beifahrerseite eine Komponente ControlComponentPrototype vorgesehen. Für das konkrete System werden dann beiden Prototypen mit (eventuell) verschiedenen Implementierungen (falls das aufgrund der Seitenumkehrung notwendig ist) für Fahrer- und Beifahrerseite instanziiert, wodurch ihre Rollen im System festgelegt sind. Unterscheiden werden sich beide Komponenten definitiv anhand ihrer Port-Instanzen, da beide jeweils über verschiedene Konnektoren mit den Sensoren auf *„ihrer“* Seite verbunden sind.

CompositionTypes werden nicht instanziiert, da sie für sich genommen kein Verhalten aufweisen, sondern lediglich Komponenten mit bestimmten Rollen enthalten. Vielmehr ergibt sich die Rolle einer Komposition implizit aus den in ihnen enthaltenen Komponenten-Instanzen.

**Einordnung von SW-C-Diagrammen in das AUTOSAR Schichtenmodell:**

Im Gesamtsystem lassen sich drei Abstraktionsschichten für Software-Komponenten unterscheiden:

1. **AtomaticSoftwareComponentTypes:** Beschreibung der Komponenten-Hierarchie der nach außen sichtbaren Eigenschaften der Komponenten (Ports mit Schnittstellenbeschreibung) und den Relationen zwischen diesen Komponenten (Konnektoren).

   Die Konnektoren beschreiben den Datenfluss zwischen Komponenten. Von technischen Details wird auf dieser obersten Schicht mithilfe des Virtual Function Bus (VFB) abstrahiert, der eine transparente Kommunikationsschicht zwischen den Komponenten bildet.

2. **InternalBehavior:** Nach der Definition von AtomicSoftwareComponent-Prototypes aus den zuvor definierten AtomicSoftwareComponentTypes auf VFB-Level erfolgt nun die Beschreibung des internen Verhaltens, d.h. der Kontrollfluss innerhalb und zwischen diesen Komponenten auf RTE- und Betriebssystem-Ebene.

Für Stimuli (z.B. eine Dienstanfrage), die an Schnittstellen von Ports einer Komponente auftreten können, werden entsprechende RTE-Events gemäß der RTE-API definiert. Die interne Reaktion der Komponente auf Stimuli besteht dann in der Ausführung eines dem RTE-Events zugeordneten Runnable in der Komponente. Ein solches atomares (schedule-bares) Code-Fragment liefert dann das geforderte Verhalten, beispielsweise indem eine Datenvariable geschrieben oder einen geforderter Dienst der Schnittstelle geliefert wird. Auf dieser Ebene werden also Festlegungen getroffenen, die Aussagen über die benötigte Rechenleistung, Speicherbedarf und Zeitverhalten der betreffenden Software- Komponenten machen und somit für die Umsetzung des RTE auf der ECU relevant sind, auf der diese Komponente im nächsten Schritt implementiert werden soll.

3. **Implementation:** Auf dieser Ebene erfolgt die Implementierung der Software-Komponente, genauer der hierin definierten Runnables, auf der Processing Unit einer ECU auf Code-Ebene.

   Dabei kann es sich um Programmiersprachen-Code (C, C++, Java, etc.) handeln, aber auch um Objektcode nach der Verwendung eines Compilers. Zur Umsetzung der geforderten Funktionen werden hier die zuvor beanspruchten RTE-API-Funktionen und System-Ressourcen verwendet.

Da an dieser Stelle die Softwarearchitektur spezifiziert werden soll, wird im Wesentlichen die erste Ebene betrachtet. Gemäß der Abstraktionssicht von AUTOSAR für diesen hier zu betrachtenden obersten Architektur-Level (VFB-Level) werden mit Softwarekomponenten (Teil-) Applikationen des Systems unabhängig von der späteren technischen Infrastruktur, d.h. den unterliegenden hardwareabhängigen Schichten, beschrieben, wodurch eine frühe Integration der Softwarekomponenten möglich ist.

Die benötigten Elemente der AUTOSAR Infrastruktur liegen nach der Implementierung der Komponenten auf einer ECU im System als Dienste des Runtime Environment (RTE) und der Basic Software (AUTOSAR-OS) vor. Das RTE stellt dabei eine spezifische Implementierung des VFB auf einem ECU dar, die abgestimmt ist auf die angeforderten Dienste und Kommunikationsprotokolle der Softwarekomponenten, die auf diese ECU zur Ausführung gemappt wurden. Die Basic Software Schicht liefert die üblichen Betriebssystem-Dienste, wie z.B. Gerätetreiber und Abstraktionen vom eingesetzten Mikro-Controller.

Somit wird durch die Struktur der SW-C implizit festgelegt, welche Anforderungen an die spätere Infrastruktur bezüglich der angeforderten Ressourcen (gegeben durch entsprechende Constraints), Dienste des Betriebssystems und der benötigten Hardware-Treiber gestellt werden.

### Hardware-Architektur und Topologie

Für die Beschreibung der elektrischen Hardware sieht der AUTOSAR Entwicklungsprozess eine Kombination aus zwei Arten von Diagrammen vor. Das ECU-Diagramm (Electronic Control Unit) definiert die Typen und Ausstattung von Steuergeräten und ihre Schnittstellen zu Bussystemen. Das Topologie-Diagramm legt die physikalische Verteilung der Hardware im System einschließlich ihrer Verschaltung fest.

#### ECU-Diagram

In diesem Diagramm wird festgelegt, welche Steuergerätetypen für die Ausführung von Softwarekomponenten im System ausgewählt werden. Für einen Steuergeräte-Typ können die elektronischen Komponenten, aus denen sie zusammengesetzt sind, sowie die Kommunikationsschnittstellen zur Einbindung dieser Elemente definiert werden.

Ein ECU-Modell besteht aus eine Menge von Hardware-Elementen (allgemeine Oberklasse HWElement), die über Hardware-Ports (HWPorts) miteinander über entsprechende Hardware- Assembly-Konnektoren verbunden sein können. An einem Hardware-Port können im Prinzip beliebig viele Verbindungen über eine entsprechende Anzahl von elektrischen Anschlüssen (Pins) realisiert werden.

Für die hierarchische Strukturierung und Gruppierung der Menge von Hardware-Elementen einer ECU können Hardware-Elementcontainer (HWElement-Container) definiert werden, die keine technische Repräsentation haben, sondern in ihrer Anwendung vergleichbar sind mit der Komposition von Software-Komponenten. Im Folgenden werden die wichtigsten Hardware-Elemente zur Spezifikation einer ECU kurz vorgestellt:

- **Processing Unit:** Spezifikation der Prozessor-Einheit des Steuergerätes. Diese stellt die eigentliche Ausführungsplattform für atomare Komponenten dar, die auf der entsprechenden ECU ausgeführt werden sollen. Die Charakterisierung der Processing Unit wird anhand von Attributen vorgenommen, auf deren Details an dieser Stelle nicht näher eingegangen werden sollen.

- **ECU Electronics:** Beschreibung der elektronischen Grundelemente des Steuergerätes wie beispielsweise die System-Clock. Auf nähere Einzelheiten soll an dieser Stelle nicht eingegangen werden.

- **Memory:** Spezifikation der im Steuergerät vorhandenen Speicherbausteine mitsamt den Verbindungsports zu anderen Hardware-Elementen und Segmentierung.

- **Sensoren und Aktuatoren:** Einer ECU können beliebige Sensor- und Aktuator-Elemente zugeordnet werden. Diese Hardware-Elemente entsprechen der technischen Repräsentation der Sensor/Aktuator Software-Komponenten, die Datenquellen bzw. -senken in der funktionalen Architektursicht spezifiziert haben. Spezialform einer Aktuator-Hardware-Einheit ist ein Display-Element (z.B. eine LED).

- **Peripherie:** Schließlich können der ECU eine Vielzahl von Peripherie-Elementen zugeordnet werden, die über einen speziellen Peripherie-Hardware-Port angebunden werden. Das sind im Wesentlichen analoge/digitale IO-Ports, AD/DA-Wandler, Pulsweitenmodulation sowie Kommunikationsspezifische und Puffer-Elemente.

An dieser Stelle sei noch mal daran erinnert, dass bei der Einbindung dieser Hardware-Elemente in ein AUTOSAR-System für die Ausführung von Softwarekomponenten vom direkten Hardware-Zugriff abstrahiert wird. Die konkrete Ansteuerung der Hardware ist im ECU-spezifischen RTE realisiert, das beispielsweise Dienste in der Basic Software zur Verfügung stellt, um auf die Peripherie-Elemente zuzugreifen.

Dieser Ansatz ermöglicht es, die Softwarekomponenten unabhängig von der ECU, auf der sie später ausgeführt werden sollen, zu spezifizieren, in dem die standardisierten Dienste der RTE-API als Abstraktionssicht auf das unterliegende System angenommen wird. Bei der späteren Implementierung des Systems müssen dann im ECU-spezifischen RTE auf einem Steuergerät und der darauf laufenden Komponenten, die benötigten Dienste vorhanden sein. Dazu gehören neben der Umsetzung des VFB, also der transparenten Kommunikation mit beliebigen anderen Komponenten im System, die Bereitstellung von benötigten Treibern, Zugriff auf den Speicher und Unterstützung des jeweiligen Mikro-Controllers durch die Basic Software. Der hier dargestellte Detaillierungsgrad zur Spezifikation von ECUs wird im Folgenden nicht weiter betrachtet und es wird von einer Standardausstattung der Steuergeräte ausgegangen.

### Topology-Diagram

Die (physikalische) Topologie eines Systems beschreibt die Auswahl und Anzahl der Hardware- Elemente (insbesondere der ECUs) und ihre Vernetzung untereinander im System über physikalische Leitungen bzw. Bussysteme sowie die Art der Verbauung, also die räumliche Verteilung der physikalischen Einheiten im Gesamtsystem. Die verwendeten Steuergeräte sind Instanzen der ECU-Typen, die im ECU-Diagramm definiert wurden und deren Kommunikationsports je nach Rolle der ECU entsprechend instanziiert werden, d.h. mit unterschiedlichen Konnektoren verbunden werden können.

Die Integration dieser Hardware-Elemente im System, d.h. die technische Umsetzung der Konnektoren zwischen Hardware-Ports (Anschlüssen) in Form von physikalischen Leitungen wird in AUTOSAR durch sog. CommunicationCluster spezifiziert. Im Bereich von automotive Steuergeräte-Netzwerken sind solche 1-n-Kommunikationswege entweder jeweils durch 1-1-Direktverbindungen (n Einzelleitungen) oder durch Bussysteme realisierbar, von denen folgende Standards als CommunicationCluster in AUTOSAR eingesetzt werden können:

- CAN (Controller Area Network)
- LIN (Local Interconnect Network)
- MOST-Bus (Media Oriented Systems Transport)
- FlexRay

Um mehrere Bussysteme mit gleichen oder verschiedenen Standards und Protokollen im System verwenden und miteinander verbinden zu können, werden zwischen zwei CommunicationsClustern Gateways in Form von PhysicalChannels definiert. Dieses ist beispielsweise notwendig, um *„globale“* Sensordaten in verschiedene Teilsysteme zu verteilen, also Daten die für unterschiedliche Steuerungssysteme relevant sind.

Später werden dann über die Hardware-Ports von Hardware-Elementen (z.B. ECUs) im System, über die sie mit anderen Einheiten über einen CommunicationsCluster verbunden sind, die entsprechenden Software-Ports (RPorts, PPorts) der darauf laufenden Softwarekomponenten umgesetzt. Dazu wird der Datenfluss gemäß der Konnektoren zwischen zwei Ports und der zugehörigen Schnittstellen in entsprechenden Frames des verwendeten Busprotokolls umgesetzt. Später wird anhand der Fallstudie die Spezifikation einer Systemtopologie verdeutlicht.

### Systemarchitektur

Als letzter Schritt ergibt sich schließlich die Systemarchitektur durch Integration der Software- Architektur und der Hardware-Architektur. Für jede atomare Softwarekomponente wird nun entschieden, auf welcher ECU sie implementiert und ausgeführt wird (Funktionen-Mapping), siehe Abbildung 2.1. Durch die Verteilung der Software-Funktionen ergeben sich entsprechende Anforderungen an das System, die bei der endgültigen Implementierung der Gesamtfunktionalität zu berücksichtigen sind:

- Aus der Verteilung von miteinander kommunizierenden Komponenten gemäß ihrer Ports und instanziierten Konnektoren lässt sich die Kommunikationsmatrix für das System ableiten.

- Anhand der (Bus-) Vernetzung im System und dem notwendigen Datenverkehr zwischen Softwarekomponenten gemäß ihrer Verteilung auf den physikalischen Leitungen im System ergeben sich die entsprechenden System-Signale. Dabei wird unterschieden zwischen End-to-End-Signalen (von Port zu Port) und Cluster-Signalen zwischen den Gateways im System je nach Umsetzung des Kommunikationsweges im Bordnetz. Je nach verwendetem Kommunikationsprotokoll (Bus-Standard) erfolgt dann das Datenmapping in ein entsprechendes Frame-Format für die Übertragung über das Bus-System.
- Aus den Anforderungen und Constraints einer Komponente (Hardware-Zugriff, Speicherbedarf, Kommunikation, etc.), von denen zunächst durch den VFB abstrahiert wurde, ergeben sich nun für deren Implementierung auf einer ECU entsprechende Vorgaben für die zugehörige RTE-Variante (Treiber, Dienste der Basic Software, etc.).

**Systems-Diagram**

In AUTOSAR wird die Beschreibung der Integration von Software-Komponenten in die Hardware-Architektur (Mapping) durch System-Diagramme beschrieben. Die genauen Bestandteile dieser Sicht, die im Wesentlichen die in den vorher genannten Diagrammen definierten Strukturen referenzieren, werden nachfolgend im Detail anhand der Fallstudie beschrieben. Mit Hinblick auf die Evaluierung der sich nach diesem Schritt ergebenden Architekturvarianten lässt sich nun eine Vielzahl von Aussagen über das System ableiten, eventuell unter Betrachtung konkreter Anwendungsszenarien:

- Die zu erwartende Auslastung der Processing Unit, des Speichers und anderer Ressourcen der ECUs gemäß der darauf laufenden Software-Komponenten.
- Durch die Kommunikationsmatrix lassen sich Aussagen über die zu erwartende Buslast machen.
- Die Hardware-Kosten ergeben sich durch die Auswahl und Anzahl der verwendeten Hardware-Elemente.
- Die Software-Kosten ergeben sich aus Komplexität und Anzahl umzusetzender Softwarekomponenten.
- Anhand der verbauten Leitungen, Anschlusszahlen von Hardware-Elementen, Dimensionierung und Auslastung von Ressourcen, etc. können Aussagen bezüglich der Skalierbarkeit und Erweiterbarkeit des Systems gemacht werden.

Abschließend soll anhand der Darstellung in Abbildung 2.4 kurz (vereinfacht) veranschaulicht werden, wie aus Sicht der AUTOSAR Struktur der Messwert eines Sensors im System bis zur Verarbeitung in eine Softwarekomponente gelangt. Ein physikalisches Signal von der Umgebung (z.B. die Außentemperatur)

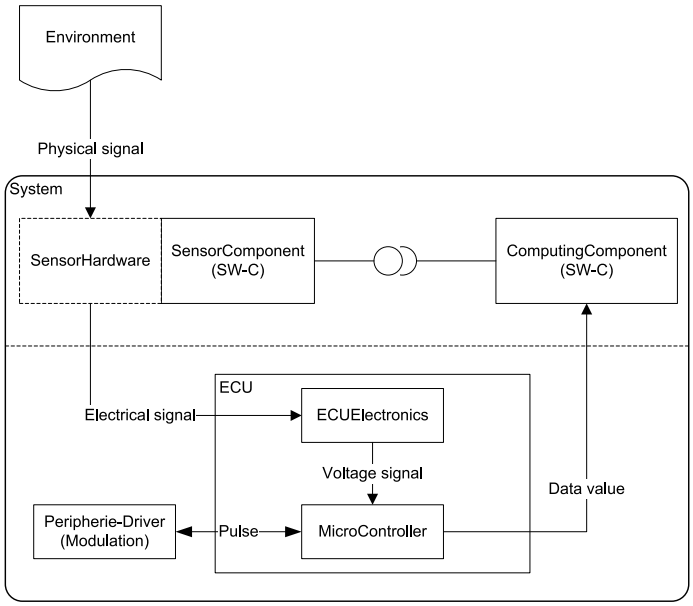


**Abbildung 2.4:** Verarbeitung eines Sensorsignals

wird durch eine Sensorhardware (z.B. Außenthermometer) registriert. Für die Weiterverarbeitung des Signalwertes in einem Steuergerät wird es zunächst in ein elektrisches Signal umgewandelt, das von einem speziellen Teil des Steuergerätes, der ECUElectronics, aufgenommen wird. Hier wird das Signal auf einen für den Mikrocontroller geeigneten Spannungspegel gebracht und eventuell durch ein entsprechendes Peripherie-Gerät eine Pulse-Modulation vorgenommen. Der Mikrocontroller interpretiert nun den Wert und setzt ihn in den entsprechenden Datenwert und Typ zur Verwendung im Softwaresystem um.

Auf Ebene der Software wird von diesem Ablauf abstrahiert, also die Sensor-Hardware als Datenquelle repräsentiert, die entsprechenden Werte direkt über einen Konnektor dem Port der verarbeitenden Komponente liefert. Für den umgekehrten Fall, d. h. die Erzeugung eines Steuersignals für einen Aktuator durch eine Softwarekomponente, ergibt sich ein entsprechender Ablauf.

## 2.5 Schichtenaufbau der Software-Architektur

Die Konzeption von AUTOSAR sieht eine klare Trennung von Applikations-Software und der ECU-Hardware vor, wie die Architekturentwurf bereits auf-

zeigte. Prinzipiell soll jede Softwarekomponente jeder unterstützen Hardware zugewiesen werden können. Der Schichtenaufbau der Software-Architektur gewährleistet diesen Anspruch mit mehreren Abstraktionsebenen.

Abbildung 2.5 zeigt die Schichten der Software-Architektur mit funktionalen Unterteilungen. Zwischen der Ebene AUTOSAR Runtime Environment (RTE) und der untersten Ebene mit der Beschriftung Mikrocontroller, die eine Hardware-Schicht darstellt, liegen die Schichten der Basis-Software (BSW).

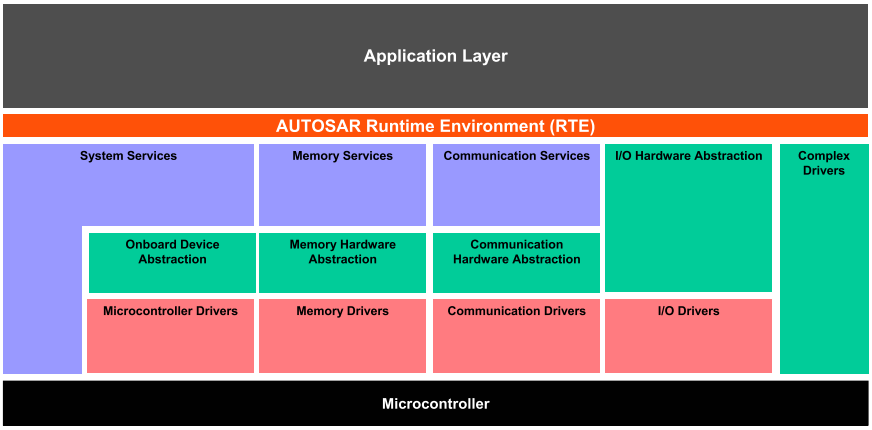


**Abbildung 2.5:** Schichten der Software-Architektur [AUT07b]

### Mikrocontroller Abstraction Layer

Zugriffe auf die Mikrocontroller-Hardware finden über die in vier Bereiche unterteilte Schicht Mikrocontroller Abstraction statt. Die Bereiche sind mit Mikrocontroller Drivers, Memory Drivers, Communication Drivers und I/O Drivers gekennzeichnet. Wie aus den Namen ersichtlich ist, handelt es sich bei diesen Komponenten um Treiber, die im Falle des verwendeten TriCore-Boards von der Firma Infineon bereitgestellt werden. Die Treiber greifen direkt auf interne Peripheriebausteine des Mikrocontrollers oder auf externe Bausteine zu, die in den Adressbereich der CPU eingeblendet werden. Die Treiber besitzen zu den darüber liegenden Schichten der Basis-Software standardisierte Schnittstellen. Auf diese Weise wird gewährleistet, dass Module der Basis-Software über der Treiberebene unabhängig vom Mikrocontroller erstellt werden können.

### ECU Abstraction Layer

Auf einem Steuergerät befinden sich im allgemeinen neben dem Mikrocontroller zusätzliche Peripheriebausteine. Die ECU Abstraction Ebene enthält ei-

nerseits Module, die interne Bausteine des Mikrocontrollers über deren Treiber ansteuern, und andererseits Module, die selbst Treiber für Bausteine außerhalb des Mikrocontrollers darstellen. Die Ebene enthält die Blöcke Memory-, Communication- und I/O Hardware Abstraction sowie Onboard Device Abstraction. Über diese Blöcke stellt die ECU Abstraction Schicht den darüber liegenden Modulen Schnittstellen für den Zugriff auf Peripheriebausteine und andere Hardware-Komponenten des Steuergerätes zur Verfügung. Ob sie sich inner- oder außerhalb des Mikrocontrollers befinden, spielt keine Rolle. Damit kann die Unabhängigkeit höher gelegener Module der Basis-Software vom Aufbau des Steuergerätes sichergestellt werden.

#### Services Layer

Die Services Ebene stellt elementare Dienste für Netzwerkkommunikation und -management des Fahrzeugs für die Diagnose und die Kontrolle über den Betriebszustand des Steuergerätes bereit. Weiterhin bietet sie Funktionen eines Betriebssystems und Zugriff auf Speicherkomponenten, wie z.B. NVRAM. Teilweise können Module der Services-Schicht Mikrocontroller- oder Steuergeräte-spezifisch angelegt sein. Diese Schicht ist im überwiegenden Anteil die höchstgelegene im Bereich der Basis-Software. Für die darüber liegende RTE stellt sie die Unabhängigkeit von Mikrocontroller und Steuergeräte-Hardware sicher. Im Falle von I/O-Zugriffen spricht die RTE direkt die ECU Abstraction Schicht an, durch die ebenfalls eine Hardware-Unabhängigkeit gegeben ist.

#### Complex Drivers

Über Complex Drivers besteht die Möglichkeit, nicht standardisierte Hardware ansprechen zu können. Es könnte sich dabei beispielsweise um sehr neue oder selbst entwickelte Bausteine handeln. Komplexe Sensorabfragen oder Steuerungsaufgaben können hier implementiert werden. Auch bei der Migration bestehender Steuergeräte-Software in eine AUTOSAR-Umgebung können Complex Drivers für eine effiziente Adaption genutzt werden. Innerhalb der Complex Drivers kann direkt auf Mikrocontroller oder Hardware zugegriffen werden. Elementare Bedingung für den Einsatz ist das Vorhandensein von AUTOSAR-konformen Schnittstellen, die sich von der RTE oder Modulen der Basis-Software ansprechen lassen.

#### Runtime Environment (RTE)

Die AUTOSAR-Laufzeitumgebung RTE repräsentiert den Virtual Functional Bus (VFB) für die Softwarekomponenten. Die RTE sorgt dafür, dass die Soft-

ware-Komponenten die erforderliche Kommunikationsstruktur vorfinden. Über die RTE tauschen die Softwarekomponenten Daten aus. Werden Daten mit Softwarekomponenten ausgetauscht, die auf anderen Steuergeräten liegen, nutzt die RTE die externe Vernetzung. Erfolgt der Austausch mit Software-Kompo-nenten auf dem gleichen Steuergerät, kann die RTE einfache Funktionsaufrufe durchführen. Für die Softwarekomponenten ist nur die Bereitstellung der Kommunikationskanäle durch die RTE relevant, die Umsetzung der Kommunikation sollte keine Rolle spielen.

Um den Programm-Code des Steuergerätes möglichst effizient zu erstellen, wird die RTE spezifisch für jedes Steuergerät mit entsprechenden Tools generiert und unterscheidet sich daher von Steuergerät zu Steuergerät.

## 2.6 Module der Basis-Software

Innerhalb der Schichten und Blöcke der Basis-Software befinden sich einzelne Module. Abbildung 2.6 zeigt die Zuordnung einiger Software-Module im Schichtenaufbau. Die meisten Module besitzen Schnittstellen in beide vertikalen Richtungen, d.h. zur darüber- und zur darunter liegenden Ebene.

Bestimmte andere Richtungen für Steuerung und Datenaustausch sind auch zugelassen. Eine detaillierte Dokumentation der Software-Schichten und Interaktionsmöglichkeiten zwischen Module ist unter [AUT07b] zu finden. Exemplarisch für die Signalpfade von der RTE durch die Basis-Software zur Hardware-Ebene, werden im Folgenden die elementaren Bereiche für Kommunikation und I/O-Zugriff kurz beschrieben.

#### Pfad durch die Kommunikationsmodule der Basis-Software

Bei der Vernetzung von Steuergeräten oder beim Datenaustausch mit der Außenwelt werden die Kommunikationsmodule der Basis-Software verwendet. Abbildung 2.7 zeigt den Bereich der Basis-Software für die Kommunikation. In den vier Schichten RTE, Services, ECU Abstraction und Mikrocontroller Abstraction sind jeweils die einzelnen Module und ihre Verbindungen zu erkennen.

Im Rahmen der später vorgestellten Fallstudie wurden u.a. die Module eingesetzt und konfiguriert, die für die Kommunikation über CAN benötigt werden. Der entsprechende Signalpfad wird durch die eingezeichnete rote Linie angedeutet. Bei den Signalen handelt es sich letztendlich um Datenpakete, die in beide Richtungen ausgetauscht werden. Wird von Softwarekomponenten ein Datenpaket über CAN gesendet, so gelangt es über die RTE, COM-Modul, PDU Router, CAN Transport Layer (CAN TP), CAN Interface und CAN Driver schließlich zur physikalischen Transportebene. Bei empfangenen Datenpaketen läuft der Pfad in entgegengesetzter Reihenfolge.

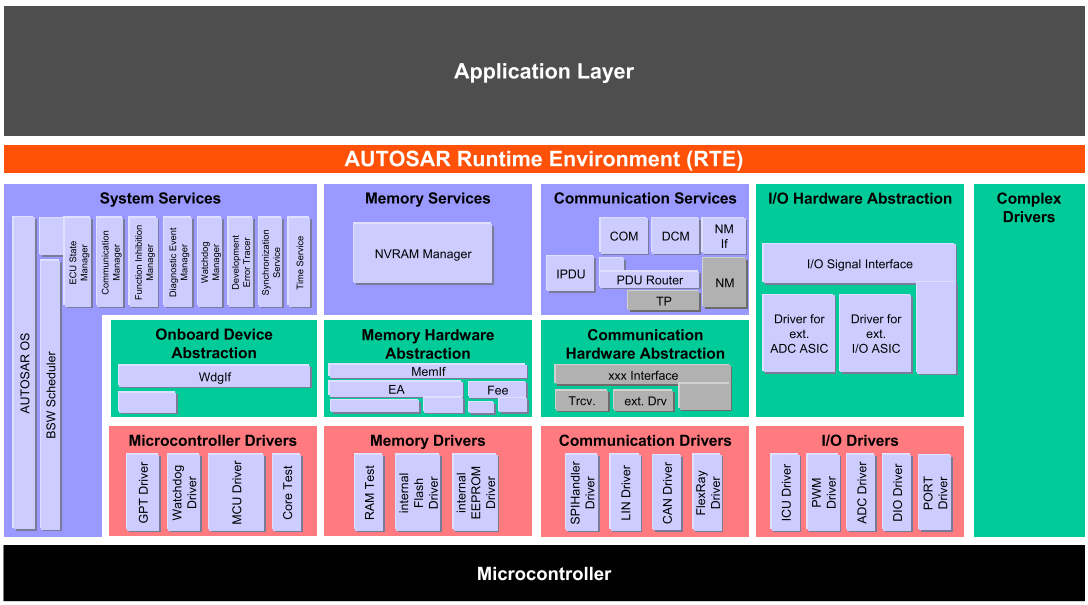


**Abbildung 2.6:** Module der AUTOSAR-Basis-Software [AUT07b]

### Protocol Data Units

Bei der Weitergabe der Daten über die verschiedenen Ebenen wird den Paketen auf der Senderseite ein Kontrolldatenblock hinzugefügt, der z.B. Informationen über Sender und Empfänger enthalten kann. Das Datenpaket, das von einer höheren Schicht an eine darunter liegende Schicht transportiert wird, wird Service Data Unit (SDU) genannt. Die hinzugefügten Kontrolldaten tragen die Bezeichnung Protocol Control Information (PCI). Das aus SDU und PCI zusammen gesetzte Datenpaket heißt Protocol Data Unit (PDU). Die PDU werden in Abhängigkeit von der Schicht, in der sie transportiert werden, unterschieden. Eine PDU im Bereich Communication Service ist eine Interaction Layer PDU (I-PDU) und eine PDU auf der Transport Protocol (TP) Ebene heißt Network Layer PDU (N-PDU). In den unteren Schichten der Communication Abstraction und der Mikrocontroller Abstraction trägt sie die Bezeichnung Data Link Layer PDU (L-PDU) [AUT07b]

**COM-Modul** Das COM-Modul stellt einen Netzübergang (engl. gateway) dar und ist zwischen RTE und PDU Router angeordnet. Es stellt der RTE eine bidirektionale Schnittstelle für AUTOSAR-konforme Signale zur Verfügung. Das COM-Modul übernimmt die Paketierung von Signalen, die versendet werden sollen, in I-PDUs. In der anderen Richtung wandelt es

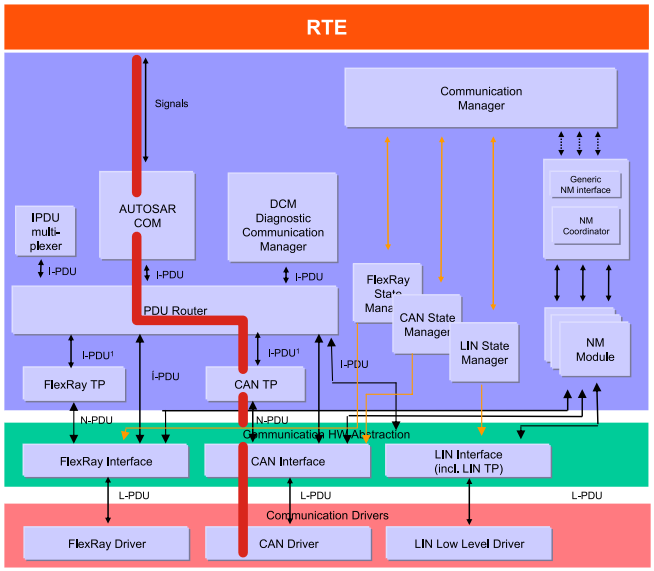


**Abbildung 2.7:** Signalpfade durch die Kommunikationsmodule der Basis-Software [AUT07l]

empfangene I-PDUs in AUTOSAR-konforme Signale um. Weitere Funktionen sind z.B. das Überwachen und Filtern von eingehenden Signalen, die Anpassung der Byte-Reihenfolge, die Vorzeichenerweiterung und das Setzen von Initialwerten. Die Dokumentation des COM-Moduls ist unter [AUT07i] zu finden.

**PDU Router** Der PDU Router dient zur Durchleitung von I-PDUs zwischen verschiedenen Modulen im Communication Service Bereich und auch zur Communication Abstraction Ebene. Abbildung 2.7 zeigt den zentralen Sitz des PDU Routers mit seinen Verbindungswegen. Die Liste der Module umfasst das COM-Modul, den I-PDU Multiplexer, die Transport Protocol Module für CAN und FlexRay (CAN TP, FlexRay TP), die CAN-, FlexRay- und LIN-Interface Module und den AUTOSAR Diagnostic Communication Manager (DCM). Der PDU Router identifiziert das Ziel einer I-PDU anhand der darin enthaltenen ID und einer statischen Konfigurationstabelle. Die Pakete werden beim Weiterreichen vom PDU Router nicht modifiziert. Der PDU Router ist in [AUT07l] beschrieben.

**CAN Transport Protocol Module (CAN TP)** Das CAN-TP-Modul befindet sich zwischen PDU Router und CAN Interface. In erster Linie hat es beim Senden die Aufgabe, I-PDUs, die größer als acht Byte sind, auf mehrere CAN-Pakete zu segmentieren. In der umgekehrten Richtung setzt es größere Datenpakete aus mehreren CAN-Paketen wieder zusammen. Die Entschei-

dung, ob das Transport Protocol angewendet werden muss oder nicht, wird im PDU Router oder im CAN Interface getroffen. Das CAN-TP-Modul ist in [AUT07h] spezifiziert.

**CAN Interface** Das CAN Interface bietet den Modulen der Communication Service Schicht eine einheitliche Schnittstelle für verschiedene CAN-Controller und CAN-Transceiver an. Das CAN Interface ist dabei an die spezielle Mikrocontroller- und Steuergeräte-Hardware angepasst. Abbildung 2.8 zeigt ein Beispiel für eine Konfiguration des CAN-Systems.

In dem Beispiel besteht das CAN-System des Steuergerätes aus im Mikrocontroller integrierten CAN-Controllern, einem zusätzlichen CAN-ASIC, das an einer SPI-Schnittstelle angeschlossen ist, und einem CAN-Transceiver, der über I/O-Pins angesprochen wird. Über das CAN Interface und spezielle Treiber für ASIC und Transceiver wird dieser spezielle Aufbau des Steuergerätes gegenüber den höheren Software-Schichten abstrahiert. Unter [AUT07g] ist das CAN Interface dokumentiert.

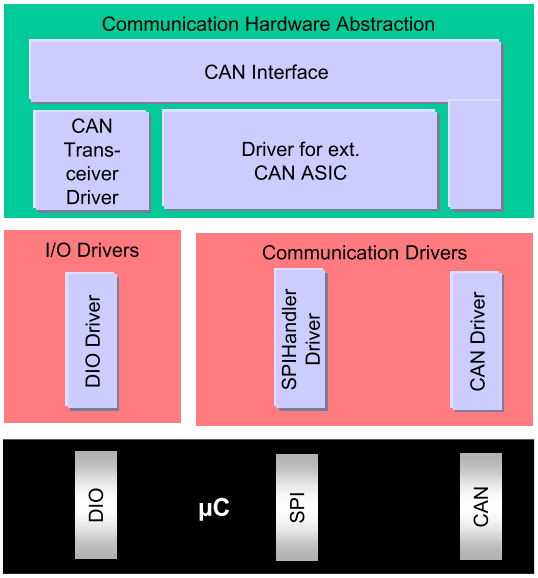


**Abbildung 2.8:** Beispiel für CAN-System und Interface [AUT07b]

**CAN Driver** Der CAN-Treiber befindet sich an unterster Stelle des Kommunikationspfades. Er greift auf die Hardware zu und stellt dem darüber liegenden CAN Interface eine Hardware-unabhängige Schnittstelle zur Verfügung. Das CAN Interface Modul hat als einziges Zugriff auf den CAN-Treiber. Der CAN-Treiber kann auch den Zustand und das Verhalten der CAN-Controller verändern. Für mehrere in einem Baustein befindliche CAN-

Controller kann ein einziger Treiber eingesetzt werden. Die Spezifikation des CAN-Treibers ist unter [AUT07f] zu finden.

### I/O Hardware Abstraction

In nahezu jedem Steuergerät werden Port-Pins des Mikrocontrollers genutzt. In vielen Fällen werden Analog-Digital-Wandler oder Hardware-Timer eingesetzt. Wie diese elementaren Anforderungen unter AUTOSAR mit Modulen der Basis-Software umgesetzt werden können, ist in [AUT07k] beschrieben.

Abbildung 2.9 zeigt die Konzeption für I/O-Zugriffe von den Software-Komponenten, die durch RTE und Basis-Software zur Steuergeräte-Hardware gelangen. Zwischen RTE und den dargestellten Treibern der Mikrocontroller Abstraction Ebene liegt der Bereich I/O Hardware Abstraction. Zur Mikrocontroller Abstraction Ebene für I/O-Zugriffe gehören die ICU-, ADC-, PWM-, DIO-, Port- und GPT-Treiber, die im Folgenden kurz erläutert werden.

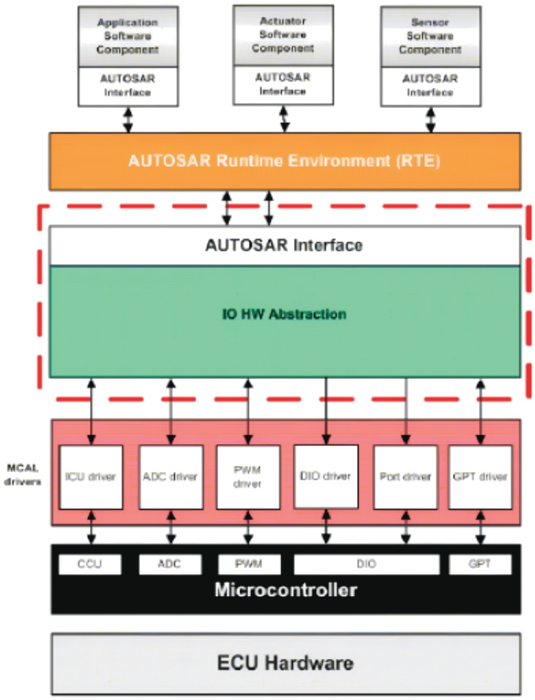


**Abbildung 2.9:** Verlauf von I/O Zugriffen durch die Schichten der Basis-Software [AUT07k]

**ICU Driver** Der ICU-Treiber stellt eine Schnittstelle zur Input Capture Unit (ICU) bereit. Diese Hardware wird z.B. zur Flankenerkennung, zum Zählen

von Pulsen, zur Frequenzmessung, zur Demodulation von PWM-Signalen und zur Bestimmung des Tastverhältnisses von Signalen genutzt. Erkannte Flanken können z.B. Interrupts auslösen oder den Mikrocontroller „aufwecken“. Die Spezifikation des Treibers ist unter [AUT07d] zu finden.

**ADC Driver** Der ADC-Treiber kontrolliert einen oder mehrere im Mikrocontroller integrierte Analog Digital Converter (ADC). Der Treiber kann den Wandler starten und stoppen, die Trigger-Quelle schalten, Status und digitalisierten Wert abfragen sowie Benachrichtigungssignale auslösen. In [AUT07e] ist der ADC-Treiber spezifiziert.

**PWM Driver** Der PWM-Treiber dient zur Ansteuerung der Hardware zur Pulsweitenmodulation (PWM, engl. Pulse Width Modulation). Mit dem Treiber kann die Periodendauer und das Tastverhältnis des Ausgangssignals eingestellt werden. Die Beschreibung des PWM-Treibers liefert Dokument [AUT07n].

**Port Driver** Der Port-Treiber dient zur Konfiguration und Initialisierung der Pins und Ports des Mikrocontrollers. Die Pins und Ports können im Betrieb für verschiedene Zwecke genutzt werden, z.B. für allgemeine I/O-Zugriffe, für eine PWM, zur Analog-Digital-Wandlung, für SPI-Zugriffe oder als CAN-Schnittstelle. Der Port-Treiber ist in [AUT07m] spezifiziert.

**DIO Driver** Die Abkürzung DIO steht für Digital Input Output. Von höheren Software-Schichten erfolgen das Einlesen von Pins und das Ausgeben auf Pins über den DIO-Treiber. Ein Pin wird beim DIO-Treiber mit Channel bezeichnet. Der DIO-Treiber bietet die Möglichkeit, mehrere Pins zu Channel Groups zusammenzufassen. Damit können beispielsweise 16 Port Pins mit einer einzigen Abfrage synchron als Wort eingelesen oder ausgegeben werden. Die angesprochenen Pins müssen mit dem Port-Treiber konfiguriert werden. Die Spezifikation des DIO-Treibers ist in [AUT07j] zu finden.

**GPT Driver** Der Treiber für General Purpose Timer (GPT) stellt die Funktionen dieser Hardware-Komponente dem Betriebssystem (OS) oder Modulen der Basis-Software zur Verfügung. Er ist eine Alternative zum OS Alarm Service für Aufgaben im Kurzzeitbereich (typisch 5 µs bis 5 ms). Der GPT-Treiber ist in [AUT07p] spezifiziert.

### Richtlinie zur Implementierung der I/O Hardware Abstraction

Die Spezifikation der I/O Hardware Abstraction bezieht sich auf den Bereich zwischen der RTE und den Treibern. In Abbildung 2.9 ist dieser Bereich rot

umrandet und trägt die Bezeichnung *„AUTOSAR Interface / IO HW Abstraction“*. Es wird ausdrücklich darauf hingewiesen, dass die Spezifikation nur als Richtlinie aufzufassen ist. Der Block I/O Hardware Abstraction soll bei der Implementierung nicht als einzelnes Modul angesehen werden, sondern als Raum für mehrere zu implementierende Module. Die AUTOSAR-Spezifikation für die I/O Hardware Abstraction Schicht will darin enthaltene Module ausdrücklich nicht standardisieren. Es werden lediglich die funktionalen Schnittstellen zu anderen Modulen definiert.

Die I/O Hardware Abstraction Schicht soll den Zugriff auf die genannten Hardware-Komponenten vom Steuergerät unabhängig machen, so dass bei der Entwicklung der Softwarekomponenten die physikalische Implementierung nicht berücksichtigt werden muss. Zur Realisierung schlägt die Spezifikation vor, die I/O-Signale eines Steuergerätes den Modulen der I/O Hardware Abstraction als Ports zuzuordnen und diese Module als Softwarekomponenten aufzufassen.

## 2.7 Betriebswirtschaftliche Aspekte in AUTOSAR

AUTOSAR versucht Prozesse während der Entwicklung eines automotiven Systems zu standardisieren. Durch den Anstieg des Vernetzungsgrades in automotiven Steuergerätenetzwerken, steigt auch die Anzahl der Beteiligten bei der Entwicklung solcher Systeme. Zur Beherrschung dieser neuen Herausforderungen ist eine kooperative Zusammenarbeit mehrerer Hersteller und Zulieferer unerlässlich.

Im Hinblick auf die Praxis, wird die AUTOSAR-Methodik wie folgt angewandt, um eine kooperative Zusammenarbeit zu ermöglichen. Ein Hersteller definiert am Anfang des Entwicklungsprozesses einer neuen Fahrzeug-Variante eine Beschreibung für die Software-Komponenten und stellt diese als XML-Datei seinen Software-Zulieferern zur Verfügung. In dieser Beschreibung sind die Spezifikationen für die Anwendung, die Aktuator- und Sensor-Software-Komponenten sowie die Kommunikationsschnittstellen enthalten. Diese vordefinierten Schnittstellen ermöglichen eine Zulieferer-übergreifende Entwicklung sowie die Integration neuer Funktionen im fortgeschrittenen Entwicklungsprozess [AUT09b].

Diese Zusammenarbeit setzt jedoch voraus, dass die Systemelemente wiederverwendbar und austauschbar sind. Dies gilt insbesondere beim Entwurf der unterschiedlichen Software-Module. Die Wiederverwendung und Austauschbarkeit dieser Modulen erhöht die Flexibilität in der Zusammenarbeit zwischen Herstellern und Zulieferern. Dennoch muss dabei die Wettbewerbsfähigkeit der einzelnen Zulieferer durch den Schutz des Knowhows garantiert werden.

### Wiederverwendung

Durch die Anpassung einer Software-Komponente an nicht-allgemeine Präferenzen, verliert eine Komponente grundsätzlich an Wiederverwendbarkeit. Dennoch ist es nicht rentabel für jede neue Aufgabe eine spezielle Lösung zu entwickeln. Ziel der Entwicklung ist es also, zwischen der Wiederverwendbarkeit und Spezialisierung von Software-Komponenten, einen geeigneten Kompromiss zu finden.

AUTOSAR ermöglicht durch die Standardisierung der Anwendungsschnittstellen eine schnittstellenorientierte Entwicklung von wiederverwendbarer Software-Komponenten. Diese Spezifikation ist dabei flexibel und dient ausschließlich als Vorlage. Somit wird die Innovationsfähigkeit und der Wettbewerb bei der automotiven Software-Entwicklung nicht gebremst.

Durch die Festlegung der Schnittstellen, kann der Software-Entwickler auf die Herausgabe des Quellcodes verzichten. Die entwickelten Software-Module lassen sich in Binärform (kompilierter Programm-Code) in andere Module integrieren (vgl. Plugins bei PC-Systemen). Infolgedessen bleiben die oftmals aufwendigen Algorithmen der Software im Verborgenen und das geistige Eigentum des Entwicklers geschützt. Dies ermöglicht die Wahrung des Knowhows des Entwickler und fördert so dessen Wettbewerbsfähigkeit [KF08].

### Austauschbarkeit

Wiederverwendung und Austauschbarkeit ähneln sich in Ihren Ansätzen sehr stark. Im Gegensatz zur Wiederverwendung einer Software-Komponente, beschreibt die Austauschbarkeit die Wiederverwendung des Umfeldes dieser Komponente. So kann ein Hersteller zwischen unterschiedlichen Software-Lösungen verschiedener Zulieferer wählen. Andererseits besteht für die Zulieferer die Möglichkeit, die gleichen Software-Module unterschiedlichen Herstellern anzubieten [KF08].

## 2.8 AUTOSAR Releases

Im Laufe der Entwicklung des AUTOSAR-Standards seit Sommer 2002, erschienen bereits eine Vielzahl von Auflagen (Releases) des Standards. In der Grafik 2.10 [KF08] ist eine zeitliche Einordnung der AUTOSAR-Releases und weiterer wichtiger Meilensteine abgebildet.

Das Release 1.0 enthielt bis auf einige Teile der Basis-Software noch wenig Inhalte und war vorrangig als Konzept anzusehen. Dieses Konzept sollte zeigen, dass die erarbeiteten Ideen in der Praxis verwendbar sind.

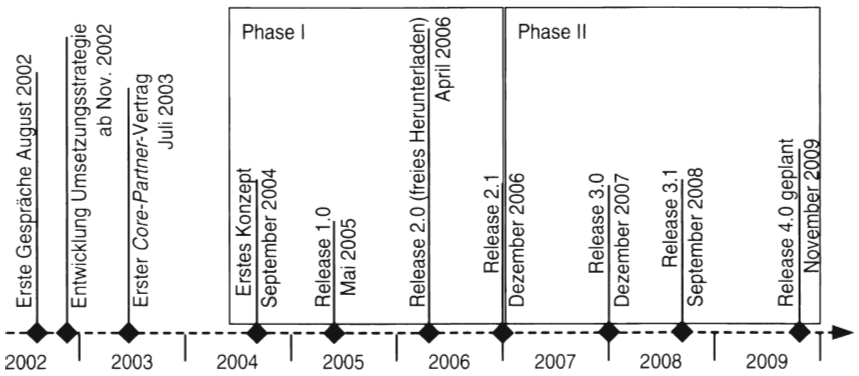


**Abbildung 2.10:** Zeitliche Einordnung der AUTOSAR-Releases

Ein wesentlicher Meilenstein wurde mit dem Release 2.0 erreicht. Diese Auflage des Standards enthielt zum ersten Mal das Konzept der AUTOSAR Laufzeitumgebung (RTE) sowie weitere Basis-Software-Module. Des Weiteren wurde der Standard erstmals der Öffentlichkeit zur Verfügung gestellt. Im Rahmen des Release 2.1 wurden kleinere Verbesserungen an den Spezifikationen vorgenommen und das vormalige Release demnach abgerundet.

Mit dem Release 3.0 erschien Ende 2007 eine konsequente Weiterentwicklung der bisherigen Spezifikationen. Insbesondere wurde eine Verbesserung der Werkzeugunterstützung durch die Einführung eines Basis-Software-UML-Modells ermöglicht. Im Zuge der Vorbereitungen auf die zukünftige Auflage des Standards (Release 4.0), wurden die Konformitätstests für Modellierungswerkzeuge vorbereitet.

Eine wiederum kleine, aber wichtige Erweiterung entstand im Zuge des Release 3.1. In diesem Release wurde die Einbindung von standardisierten Diagnosefunktionen ermöglicht. Diese Diagnosefunktionen basieren auf dem, in Amerika gesetzlich vorgeschriebenen, OBD-II-Standard. AUTOSAR erweiterte dadurch seine internationale Einsetzbarkeit [KF08].

Aktuell umfassen die Spezifikationen des Standards rund 8000 Seiten und sind im Internet frei verfügbar [AUT07a]. Die weiteren Ausführungen in diesem Bericht beziehen sich auf das zu Begin der Fallstudie verfügbare AUTOSAR Release 2.1.

# 3 Fallstudie Komfortsystem

## 3.1 Aufbau und Demonstrator

Es wurde exemplarisch eine Fallstudie an einem typischen modernen PKW-Komfortsystem umgesetzt, die an dieser Stelle kurz eingeführt werden soll. Der in diesem Rahmen entstandene Demonstrator umfasst folgende vier, über CAN vernetzte Steuergeräte:

- HMI-Steuergerät (Display und Statusanzeige, Komforteinstellungen, ...)
- Türsteuergerät (Fensterheber, Spiegelverstellung, Spiegelblinker, ...)
- Zentralverriegelungssteuergerät (Funkmodul, Schlossaktorik, Türkontakte, ...)
- Alarmanlagensteuergerät (Innenraumüberwachung, Türüberwachung, ...)

Ein Foto des vernetzten Systems ist in Abbildung 3.1 zu sehen. Der Demonstrator wurde bewusst so entworfen, dass eine hohe Anzahl an Steuergeräte- und Systemübergreifende Softwarefunktionen und damit ein hoher Vernetzungsgrad entstehen. Weiterhin sind im Demonstrator anwendungsbedingt eine hohe Anzahl an umfangreichen Benutzerschnittstellen integriert. Für eine detailierte Aufzählung aller Funktionen des Systems wird auf [Gar09] und [Mic09] verwiesen.

Der prinzipielle Aufbau des Demonstrators ist in Abbildung 3.2 dargestellt. Jedes Steuergerät ist dabei mit einer Demonstrations- und Simulationseinheit ausgestattet. Die Demonstrationseinheit beinhaltet die Sensorik und Aktorik und ermöglicht damit die direkte Interaktion des Anwenders mit dem System. Die Simulationseinheit deaktiviert bei Bedarf die Demonstrationseinheit und simuliert dabei die Sensoren und Aktoren, gesteuert durch ein zentrales HIL-Testsystem. Die Steuergeräte werden weiterhin durch eine Restbussimulation ergänzt, die eine virtuelle Kommunikation mit dem restlichen Fahrzeug ermöglicht. Mit Hilfe des bereits erwähnten HIL-Testsystems werden im Rahmen des in diesem Beitrag später beschriebenen Prozesses, automatisiert Testfälle durchgeführt.

Durch die bereits erwähnte bewusst erhöhte Komplexität des Systems eignet sich der Demonstrator in besonderem Maße als Fallstudie.
Im Folgenden sollen die einzelnen Steuergeräte des Demonstrators kurz vorgestellt werden. Für eine ausführliche Beschreibung des mechanischen Aufbaus sowie der Realsierung der elektronischen Komponenten wird auf [Mic09] verwiesen.

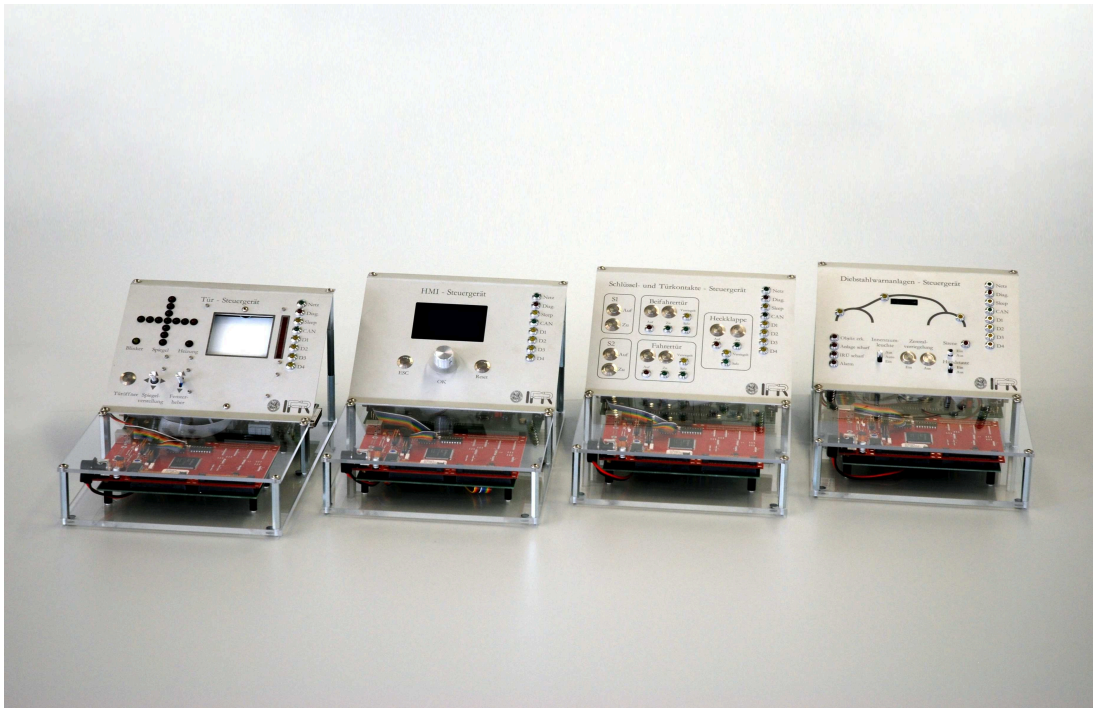


**Abbildung 3.1:** Abbildung der vier Steuergeräte

## 3.2 Funktionalitätsbeschreibung

Die umgesetzte Fallstudie deckt zusammengefasst nachfolgenden Funktionsumfang ab:

- Elektr. Fensterheber mit Einklemmschutz
- Elektr. einstellbare und heizbare Außenspiegel
- Zentralverriegelung mit SAFE für die Türen
- Zentralverriegelung (Heckklappe)
- Diebstahlwarnanlage mit Innenraumüberwachung
- Funkfernbedienung
- Zentrale Bedien- und Informationseinheit (HMI)

Prinzipiell ist die Fallstudie jedoch so ausgelegt, dass auch andere Ausstattungsvarianten möglich sein könnten. Beispielsweise soll auch eine Variante ohne

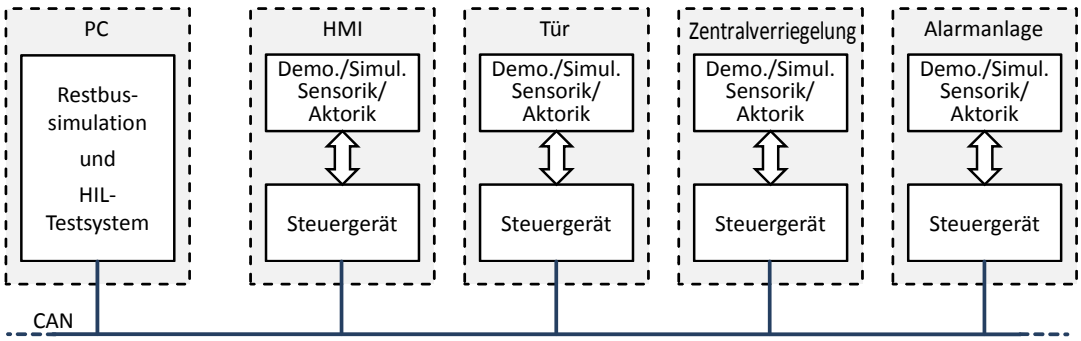


**Abbildung 3.2:** Prinzipieller Aufbau des Demonstrators

Diebstahlwarnanlage oder ohne Funkfernbedienung möglich sein. Um das Einbinden des Systems in bereits existierende Produktionsprozesse zu ermöglichen, sind weitere Eigenschaften gefordert:

- Selbsttestfunktion der einzelnen Steuergeräte
- Autarke Prüfbarkeit der Module

Nachfolgend werden die einzelnen Funktionsgruppen kurz beschrieben.

### Zentralverriegelung

Die Zentralverriegelung schließt oder öffnet auf Knopfdruck alle elektrischen Türschlösser, d.h. das Fahrzeug wird vollständig ver- oder entriegelt. Die Taster für die Zentralverriegelung sind in der Mittelkonsole vorgesehen. Ein Druck auf Taste *„Zentralverriegelung ein“* bewirkt die komplette Verriegelung des Fahrzeugs nach außen und von innen. Dieser Zustand wird auch als *„Safe“* bezeichnet. Sollte eine Tür beim Tastendruck noch geöffnet sein, werden diese verriegelt sobald der jeweilige Türkontakt das Schließen signalisiert. Analog bewirkt ein Tastendruck auf die Taste *„Zentralverriegelung auf“* die Entriegelung des Fahrzeugs. Wird die Taste *„Zentralverriegelung ein“* jedoch länger als zwei Sekunden betätigt, werden zusätzlich zur Verriegelung alle Fenster hochgefahren. Weiterhin ist eine *„Autolock“*-Funktion umgesetzt: Überschreitet das Fahrzeug eine bestimmte Geschwindigkeit, werden alle Türen automatisch verriegelt. Es handelt sich um eine Sicherheitsfunktion gegen *„Carnapping“*. Die Autolock-Funktion kann über die zentrale Bedieneinheit(HMI) ein- oder ausgeschaltet werden. Dort kann auch die Aktivierungsgeschwindigkeit verändert werden.

### Alarmanlage

Die Alarmanlage soll unbefugtes Eindringen in den Fahrgastraum mit Hilfe einer lauten Sirene verhindern. Sie kann gleichzeitig mit der Verriegelung des Fahrzeugs scharf geschaltet werden. Die Aktivierung erfolgt allerdings erst, nachdem auch tatsächlich alle Türen geschlossen wurden. Sollte dann eine Tür gewaltsam geöffnet, der (innere) Türöffner betätigt oder von der Innenraumüberwachung eine Bewegung im Fahrgastraum registriert werden, löst die Anlage den Alarm aus.

Im Falle eines Alarms wird die Sirene eingeschaltet und die Blinker werden in schneller Abfolge an- und ausgeschaltet. Diese Alarmsignale werden nach 20 Sekunden automatisch abgestellt oder enden unmittelbar durch Aufschließen des Fahrzeugs. Wird der Alarmzustand nicht manuell beendet, erscheint eine Mitteilung über den aufgetretenen Alarm auf der Anzeige und die Alarm-Kontrollleuchte blinkt in schneller Abfolge bis eine Quittierung über die zentrale Bedieneinheit erfolgt.

Bei aktivierter Alarmanlage und Innenraumüberwachung registriert ein im Armaturenbrett angebrachter Entfernungssensor Bewegungen im Fahrgastraum. Wird eine Bewegung eindeutig erkannt, löst dies einen Alarm aus. Die Innenraumüberwachung kann über die zentrale Bedieneinheit generell ein- oder ausgeschaltet werden. Weiterhin besitzt die Alarmanlage eine *„Hundetaste“*. Mit Hilfe dieses Schalters lässt sich eine aktivierte Innenraumüberwachung schnell und bequem ausschalten, z.B. für den Fall, dass Tiere im Fahrzeug verbleiben sollen.

### Funkschlüssel

Das Fahrzeug lässt sich mit Hilfe von zwei Funkschlüsseln ver- oder entriegeln sowie sichern oder entsichern. Komforteinstellungen wie die Außenspiegelneigung können für jeden Funkschlüssel individuell gespeichert werden. Wird das Fahrzeug mit einem Funkschlüssel aufgeschlossen, wird die gespeicherte Einstellung für den Außenspiegel automatisch wieder eingestellt.

Beim betätigen der Taste *„zu“* durch den Funkschlüssel werden alle Türen verriegelt, alle Fenster automatisch geschlossen und die Alarmanlage aktiviert sofern diese Option eingestellt wurde. Sollten Beifahrertür oder Heckklappe beim Druck auf die Taste *„zu“* noch geöffnet sein, werden sie verriegelt sobald der jeweilige Türkontakt das Schließen signalisiert. Die Fahrertür wird nur verriegelt, wenn die vor dem Druck auf die Schlüsseltaste *„zu“* bereits geschlossen wurde.

Sind alle Türen korrekt verschlossen, wird dies durch dreimaliges schnelles Aufleuchten der Blinker quittiert. Ist die Alarmanlage aktiviert worden, gibt die Sirene zwei kurze Töne ab.

Beim betätigen der Taste *„auf“* durch den Funkschlüssel werden alle Türen entriegelt die Alarmanlage gleichzeitig deaktiviert bzw. ein ausgelöster Alarm abgestellt.

Zusätzlich wurde eine Sicherheitsfunktion implementiert: Wurde das Fahrzeug durch Betätigen der Taste *„auf“* des Funkschlüssels entriegelt, verstreicht aus Sicherheitsgründen eine Frist, in der die Türen geöffnet werden müssen. Sollte keine Tür innerhalb von 10 Sekunden geöffnet werden, wird das Fahrzeug automatisch wieder verriegelt.

**Zentrale Bedieneinheit - HMI**

Über die zentrale Bedieneinheit(HMI) werden wichtige Informationen, Einstellungen und Ereignisse angezeigt. Sie verfügt über ein Display mit vier Zeilen zu je 20 Zeichen, ein Navigationsrad und einer ESC-Taste. Bei dem Navigationsrad handelt es sich um einen Dreh- Drück-Schalter. Über ihn können Menüs und Untermenüs ausgewählt sowie Einstellwerte verändert werden. Mit einer Betätigung des Navigationsrads wird das ausgewählte Menü aufgerufen. Mit einer Betätigung der ESC-Taste wird von der aktuellen Menüebene auf die darüber liegende zurückgesprungen. Folgende Informationen können abgerufen werden können:

- Geschwindigkeit
- Außentemperatur

Weiterhin gibt es für folgende Einstellungen eine Statusanzeige:

- Alarmanlage wird über die Zentralverriegelung automatisch aktiviert/nicht aktiviert
- Innenraumüberwachung aktiviert/nicht aktiviert (Stellung der "Hundetaste" berücksichtigen)
- Autolock aktiviert/nicht aktiviert

Folgende Ereignisse werden ebenfalls in der Statusanzeige dargestellt:

- Alarm wurde ausgelöst (Bestätigung erforderlich)
- offene Türkontakte

Neben der Informationsdarstellung werden über die zentrale Bedieneinheit auch Einstellungen vorgenommen:

- Es kann gewählt werden, ob die Alarmanlage beim Verschließen des Fahrzeugs automatisch scharf geschaltet werden soll.

- Die Autolock-Funktion lässt sich ein- oder ausschalten. Die Aktivierungsgeschwindigkeit für Autolock lässt sich verändern.
- Die Innenraumüberwachung der Alarmanlage kann generell aktiviert oder deaktiviert werden.
- Die Verzögerungszeit bis die Innenbeleuchtung nach Schließen der Türen erlischt, lässt sich mit Sekundengenauigkeit einstellen.

Um die Einstellungen speichern zu können ist die zentrale Bedieneinheit als einziges Steuergerät mit einen nichtflüchtigen Datenspeicher ausgestattet.

#### Elektrische Fensterheber

In der Fallstudie wurden weiterhin ein elektrischer Fensterheber der Fahrertür und ein Spiegel realisiert. Hierfür wurden in die Tür zwei Tasten für das Fenster und ein Bedienfeld für die Spiegelverstellung integriert. Um eine Verletzungsgefahr oder Sachbeschädigung auszuschließen wurde ein Einklemmschutz integriert, der bei einem erkannten Widerstand das Fenster wieder um 1cm senkt. In der Demonstrationseinheit wurden das Fenster mechanisch nachgebildet. Dabei wird der Verfahrweg des Fensters durch einen Impulsgeber ermittelt. D.h. es gibt keinen Absolutpositionssensor, so dass eine Initialisierungsroutine und ein internes Modell zur Positionsbestimmung notwendig wird, wie das z.B. für Fahrzeuge wie dem Golf VI der Fall ist. Für die Spiegel sorgt weiterhin eine automatische Beheizung im Winter für eisfreie Sicht. Sie wird bei einer Außentemperatur unter +1°C automatisch für 30 Sekunden aktiviert.

#### Individuelle Einstellungen

Es sollen schlüsselabhängig, fahrerindividuelle Einstellungen gespeichert werden. Hierzu zählt speziell die Spiegelneigung. Die eingestellte Spiegelposition wird individuell für jeden Fahrer abgespeichert. Die Fahrer werden dabei über die Fahrzeugschlüssel unterschieden. Bei einem Wechsel der Fahrer bzw. der Fahrzeugschlüssel werden die Spiegel automatisch auf die entsprechende Einstellung geneigt. Die Speicherung der Einstellungen erfolgt dabei jedes Mal, wenn mit einem Schlüssel abgeschlossen wird.

#### Ausschaltverzögerung für die Innenbeleuchtung

Die Innenbeleuchtung des Fahrgastraums erlischt nach dem Schließen der Türen mit einer gewissen Verzögerung. Die Verzögerungsdauer kann über die zentrale Bedieneinheit eingestellt werden. Sofern sich der Innenbeleuchtungsschalter

in der mittleren Position befindet und mindestens einer der Türkontakte geöffnet ist, ist auch die Innenbeleuchtung eingeschaltet. Erst nachdem alle Türkontakte geschlossen sind, beginnt die Verzögerungszeit bis zum Erlöschen der Innenbeleuchtung abzulaufen. Ist die über die zentrale Bedieneinheit eingestellte Dauer verstrichen, wird die Beleuchtung automatisch ausgeschaltet. Befindet sich der Innenbeleuchtungsschalter links, ist die Beleuchtung immer aus. Steht er rechts, ist die Beleuchtung immer an.

## 3.3 Prozessmodell und Werkzeugverbund

Entwicklungsprozesse für eingebettete Systeme im Automotive-Umfeld unterliegen besonderen Bedingungen und Herausforderungen wie z.B. Sicherheit, hohe Zuverlässigkeit, geringe Kosten und hohe Robustheit, um nur einige zu nennen. Zur Umsetzung eines durchgängigen Entwicklungsprozesses hat sich das V-Modell als Prozess-Modell im Automotive-Umfeld etabliert [IAB09], [Bal98].

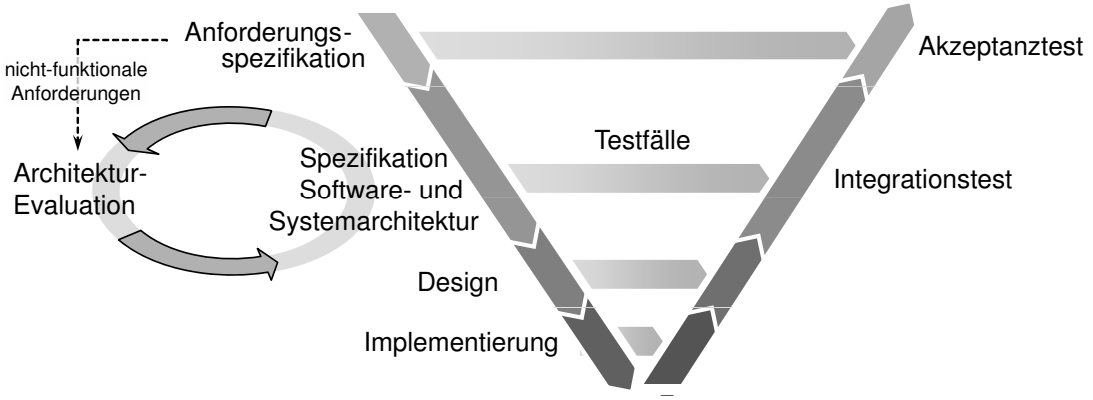


**Abbildung 3.3:** Das in der Fallstudie angewendete modifizierte V-Modell

Hinzu kommen neben der reinen Systementwicklung Aufgaben wie Projektmanagement, Konfigurationsmanagement und Qualitätssicherung, die im V-Modell nach [Bal98] nicht abgebildet sind. Gerade die für viele Automobilhersteller typische Variantenvielfalt, die sich auch in den verschiedensten Hardware- und Softwareversionen der Steuergeräte wiederfindet, stellt eine große Herausforderung auch hinsichtlich einer Versionierung dar. So ist auch weiterhin der Trend zu beobachten, dass zusätzliche Funktionalität immer mehr in Software anstatt in Hardware umgesetzt wird. Die Problematik macht die Notwendigkeit einer Initiative wie AUTOSAR deutlich. Zur Umsetzung von Systemfunktionalitäten schlägt der AUTOSAR-Standard hierbei eine ganzheitliche Vorgehensweise für die modellgestützte Entwicklung einer geeigneten Systemarchitektur bis hin zur

tatsächlichen Implementierung vor [AUT06]. So wurde auch in der zugrunde liegenden Fallstudie zunächst das V Modell entsprechend modifiziert (Abbildung 3.3). Der AUTOSAR-Standard definiert neben einem Metamodell zur Beschreibung kompletter Systemarchitekturen ein Architektur-Schichtenmodell mit standardisierten Schnittstellen, das die verschiedenen Abstraktionsebenen der Architektur integriert und zunächst von der Hardware-Plattform und Kommunikation abstrahiert. Dies steht im Kontrast zum V-Modell nach [Bal98], das zuerst mit der technischen Architektur beginnt.

Zu Begin eines Fahrzeugprojektes wird zunächst gemäß dem V-Modell mit der Analyse und Definition der Anforderungen begonnen. Die Definition der Anforderungen erfolgt auf einer abstrakten Ebene, unabhängig von einer später folgenden technischen Umsetzung. Die Anforderungen sollten dabei umfassend und möglichst widerspruchsfrei sein. Gleichzeitig können bereits hier Testfälle für die jeweilige Anforderung definiert werden. Die Erstellung und Verwaltung der Anforderungen wurde mit dem im Automotivebereich weit verbreiteten DOORS von Telelogic durchgeführt. Auf Grundlage der Anforderungen erfolgt nun, wie in Abbildung 3.3 dargestellt, mit dem bereits erwähnten Unterschied zu [Bal98], der erste Entwurf für die Spezifikation der Software- und Systemarchitektur. Die AUTOSAR-konforme Systemarchitektur selbst wurde dabei mit dem Tool SystemDesk von dSpace spezifiziert. Nach Abschluss der Systemarchitektur wurde diese einer Bewertung bzw. Evaluierung unterworfen. Auf den Schritt der Architektur-Evaluation wird in Kapitel 4.3 näher eingegangen.

Anschließend folgt der Schritt des Designs und der Implementierung. Diese wurde im Rahmen der Fallstudie modellbasiert mit einem Softwarepaket aus MATLAB/Simulink/Stateflow von The MathWorks durchgeführt. Mit dieser Entwicklungsumgebung konnte die Implementierung teilweise durch Simulationen unterstützt werden (Weiteres in Kapitel 4.4). Die Codegenerierung der einzelnen Softwarecomponents wurde letztlich mit TargetLink von dSPACE durchgeführt (Einzelheiten in Kapitel 4.4.5).

Ein entscheidender und aufwendiger Schritt ist nun die Konfiguration und Kompilierung des AUTOSAR konformen Betriebssystems. Hierfür wurde tresosECU zusammen mit dem Betriebssystem von Elektrobit eingesetzt. Nach Abschluss der Implementierung wird mit der Qualitätssicherung durch Test auf den entsprechenden Ebenen begonnen. Hierzu gehören Test der Software Komponenten, Integrationstest der Softwarekomponenten, der Systemtest sowie abschließender Abnahmetest (eine ausführliche Behandlung findet in Kapitel 4.6 statt).

Es soll an dieser Stelle besonders hervorgehoben werden, dass der gesamte linke Ast mit einer durchgehenden Toolkette durch die bereits erwähnten Tools unterstützt wird. Erst die Definition der Schnittstellen durch den AUTOSAR-Standard macht eine solche (herstellerübergreifende) Durchgängigkeit möglich. Ein solcher Toolverband ermöglicht damit einen sinnvollen Umgang mit der Kom-

plexität. Das Vorgehen wird an dieser Stelle bewusst knapp beschrieben und erst in den nachfolgenden Kapiteln ausführlicher behandelt.

## 3.4 Korrektheits- und Qualitätsanforderungen

Im Allgemein ergibt sich die Qualität von Softwaresystemen für automotive Steuergerätenetzwerke daraus, in welchem Maße die an das System gestellten Anforderungen erfüllt sind. Die Gesamtmenge der Anforderungen lässt sich dabei unterteilen in funktionale und nicht-funktionale Anforderungen.

Die funktionalen Anforderungen spezifizieren die vom System umzusetzende, durch den Kunden erlebbare Funktionalität des Fahrzeugs, beschrieben durch entsprechende Anwendungsszenarien. Immer neue Ausstattungsmerkmale, eine Vielzahl möglicher Varianten in modernen Fahrzeugprojekten sowie die Implementierung auf heterogenen Ablauf- und Kommunikationsplattformen mit spezialisierter Sensorik/Aktorik, Bussystemen etc. stellen die aktuellen Herausforderungen in der Entwicklung dar. Schon heute wird ein überwiegender Anteil der Fahrzeugfunktionen in Software realisiert. Für eine frühzeitige Überprüfbarkeit der korrekten Umsetzung funktionaler Anforderungen ist der Einsatz entwicklungsbegleitender Methoden zur Validierung der Systemspezifikation bis zur Implementierung auf der Zielplattform notwendig. Im Rahmen des V-Modells sind beispielsweise entsprechende, phasenbegleitende Testverfahren vorgesehen, darüber hinaus können Techniken wie Rapid Prototyping und HIL-Simulationen zum Einsatz kommen. Gerade bei der steigenden Komplexität moderner Fahrzeuge werden zudem modellbasierte Ansätze wie formale Verifikation und automatisierte Konsistenz- und Vollständigkeitsüberprüfungen zukünftig eine wichtige Rolle einnehmen.

Die Kategorie der nicht-funktionalen Anforderungen umfasst alle Kriterien, die sich nicht auf eine bestimmte Funktion beziehen, sondern übergeordnete Eigenschaften des Gesamtsystems betreffen. Im Folgenden werden wir diese auch extra-funktionalen Anforderungen genannten Kriterien als die eigentlichen Qualitätsanforderungen an das zu entwickelnde System bezeichnen. Im Gegensatz zu den funktionalen Anforderungen, die im Wesentlichen Kundenwünschen entsprechen, können nicht-funktionale Anforderungen durch unterschiedlichste Faktoren und Zielsetzungen der Stakeholder bestimmt werden, wie z.B.:

- Management: Kosten, Modifizierbarkeit
- Kunde: Nutzbarkeit, Robustheit
- Gesetzgeber und Standards: Sicherheit, Verfügbarkeit, Energieeffizienz
- Technische Realisierbarkeit: Prozessorauslastung, Buslast, Speicherbedarf, Gewicht

Eine Herausforderung bei der Einhaltung derartiger, projektspezifischer Qualitätsanforderungen besteht darin, dass die verschiedenen Optimierungsziele in der Regel gegenläufig sind, also ein geeigneter Kompromiss gefunden werden muss. Darüber hinaus muss eine Einschätzung der zu erwartenden Gesamtqualität des Systems möglichst frühzeitig begleitend zum Systementwurf erfolgen, um eventuell notwendige Optimierungen nicht zu aufwendig und kostspielig werden zu lassen.

# 4 Prototypische Umsetzung

## 4.1 Übersicht Prozessschritte und Tools

Das folgende Kapitel führt die in Kapitel 3.3 kurz eingeführten Schritte des modifizierten V-Modells nun detailiert aus. Dabei wird besonders auf die Methodik aber auch auf die verwendeten Tools eingegangen. Die Abbildung 4.1 zeigt eine Übersicht der eingesetzten Tools in einer schematischen Darstellung. Der Datenfluss zwischen den Ebenen und Tools ist durch die jeweiligen Pfeile skizziert. Es werden Dateien erzeugt, die von den nachfolgenden Werkzeugen verarbeitet werden. Die Pfeile sind mit dem Format der übergebenen Dateien gekennzeichnet.

Die Abbildung verdeutlicht auch, dass die durch den AUTOSAR-Standard mögliche Durchgängigkeit der Tools zum Zeitpunkt der Fallstudie noch nicht vollständig realisierbar war. Beispielsweise gab es zwischen den mit DOORS verwalteten Anforderungen und den darunter liegenden Ebenen noch keinen automatisierten Informationsaustausch. Aber auch zwischen den Tools *TargetLink* und *tresosECU* musste manuell nachgearbeitet werden. Weitere Details und Besonderheiten werden in den nachfolgenden Abschnitten näher behandelt.

## 4.2 Anforderungsspezifikation mit DOORS

Die Definition und Verwaltung der Anforderungen stellt einen elementaren und weitreichenden Schritt im Entwicklungsprozess dar, da das zu entwickelnde System sich maßgeblich daraus ableitet und daran überprüft wird. Ungenauigkeiten und Fehler auf dieser Ebene können den Entwicklungsaufwand unnötig vergrößern. Im Allgemeinen ist davon auszugehen, dass verschiedene Parteien an der Systementwicklung beteiligt sind. In einem vereinfachten Beispiel könnte ein Automobilhersteller die Anforderungen formulieren und ein oder mehrere Zulieferer die Anforderungen interpretieren und in eine technische Lösung umsetzen.

Die Anforderungen an das System unterscheiden sich in Abhängigkeit vom Personenkreis, der sie erhebt. Beispielsweise haben die Benutzer des Systems Erwartungen an den Bedienungskomfort, das Servicepersonal hat spezielle Ansprüche an Diagnose- und Wartungsfähigkeit und der Gesetzgeber wird besonderen Wert auf die Sicherheit und Umweltverträglichkeit legen. Die vielfältigen Anforderungen sollten in ihrer gesamten Bandbreite erfasst und priorisiert werden.

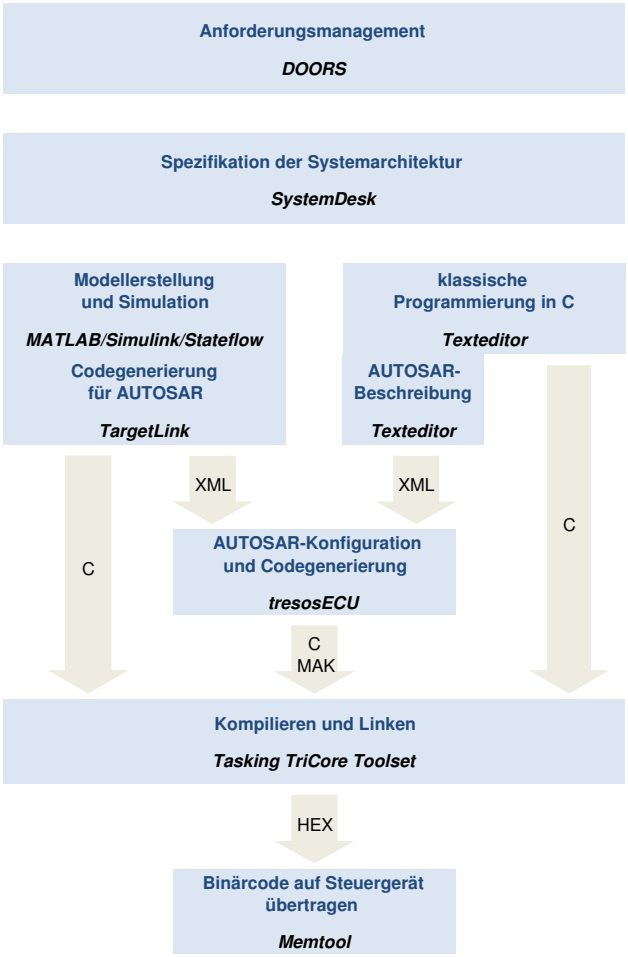


**Abbildung 4.1:** Schematische Darstellung der Toolkette

Die Anzahl und der Inhalt der Anforderungen werden sich über mehrere Zeitabschnitte verändern. So werden mit Beginn der Projektphase Anforderungen aufgestellt oder aus anderen Projekten übernommen. Im Projektverlauf können sich Änderungswünsche oder veränderte Randbedingungen ergeben, die berücksichtigt werden müssen. Schließlich fließen Fehlerbeseitigungen, Verbesserungen und neue Anforderungen im Anschluss an einen Projektzyklus ein. Das Führen einer Historie der Anforderungen einschließlich der beteiligten Personen und Begründungen bei Änderungen ist sinnvoll, um z.B. die *„Evolution“* eines Projektes nachvollziehbar zu machen.

Die Erfassung, die Überarbeitung und die Verwaltung von Anforderungen werden zum Anforderungsmanagement gezählt. Als weiterer elementarer Bereich fällt darunter auch das Nachverfolgen der Umsetzungen der Anforderungen. Anforderungen können untereinander Abhängigkeiten aufweisen. Es können Querverweise zu anderen Projekten existieren. Auch die Bezüge zu der aus den Anforderungen abgeleiteten logischen Systemarchitektur sollten in beide Richtungen dokumentiert werden. Dies gilt ebenfalls für die Bezüge zur technischen Systemarchitektur, die aus der logischen entwickelt wird.

DOORS von Telelogic stellt eine Software-Lösung für das Anforderungsmanagement dar. Mit DOORS können Anforderungen erfasst und verwaltet werden. Die Anforderungen, ihre Historie und ihre Verbindungen werden in einer Datenbank abgelegt. DOORS unterstützt die Strukturierung und das Nachverfolgen der Anforderungen. Abbildung 4.2 zeigt DOORS mit den Anforderungen des Komfortsystems. Auf der linken Seite ist die hierarchische Gliederung des Komfortsystems zu erkennen. Der Hauptbereich zeigt die Auflistung der Textblöcke mit den einzelnen Anforderungen. Die vollständigen Anforderungsdefinitionen für das Komfortsystem befinden sich in [Gar09], die Systemanforderungen in [Mic09]. Nähere Informationen zu DOORS liefern Internetseiten des Herstellers Telelogic [Tel].

DOORS oder das Anforderungsmanagement im Allgemeinen haben auch Grenzen. *„Die Analyse der Anforderungen und die Spezifikation der logischen und technischen Systemarchitektur gehören [...] zum Kernprozess der System- und Software-Entwicklung.“* [SZ06].

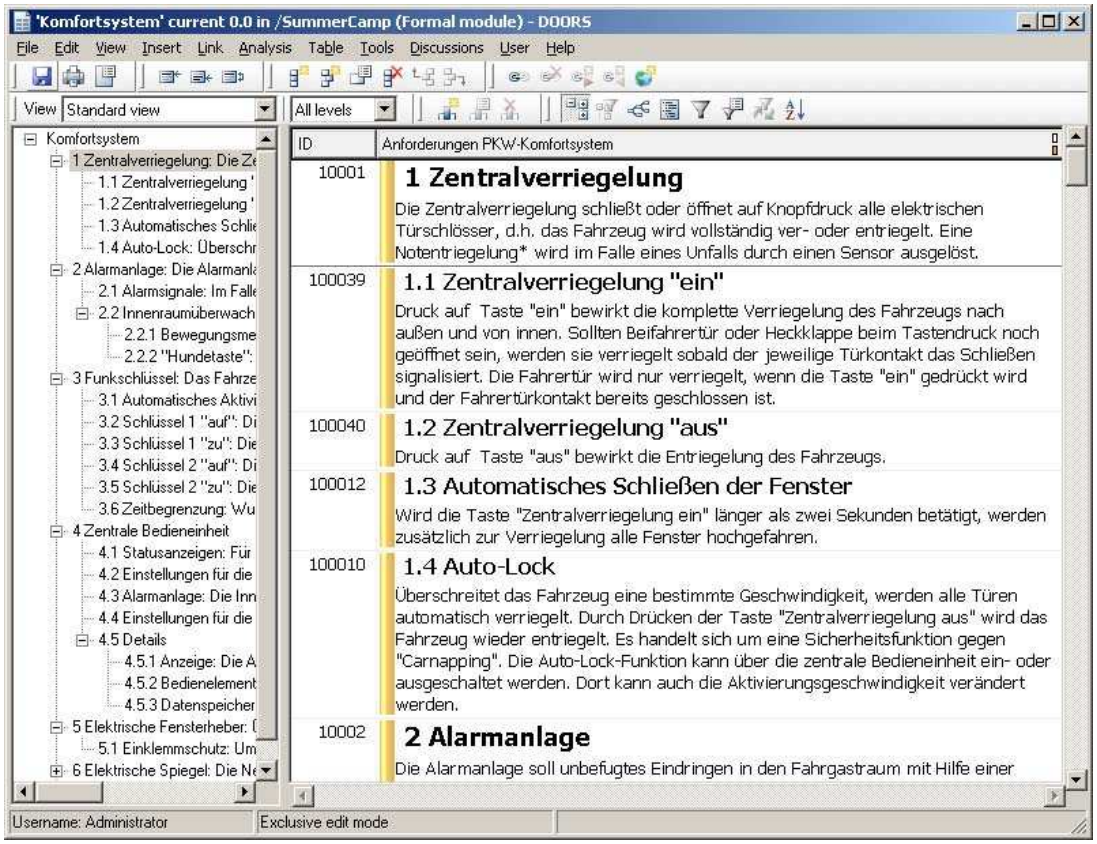


**Abbildung 4.2:** Screenshot von DOORS

## 4.3 Architektur-Entwurf und -Evaluation

Als Ausgangspunkt für die im Folgenden beschriebenen Schritte hin zu einer Software- und Systemarchitektur wird eine vom System zu erfüllende (Gesamt-) Funktionalität (Liste von Features) vorausgesetzt, wie im vorherigen Abschnitt beschrieben wurde.

Bei der Betrachtung und Bewertung verschiedener Architekturvarianten für ein Steuergerätenetzwerk zur Umsetzung dieser Anforderungen wird diese Funktionalität als invariant angesehen.

### 4.3.1 Architekturspezifikation mit SystemDesk

Zur Modellierung einer geeigneten Systemarchitektur für die Umsetzung der zuvor eingeführten Fallstudie wurde das Werkzeug SystemDesk von dSpace eingestzt. SystemDesk erlaubt neben der AUTOSAR-konformen Modellierung von E/E-Architekturen auch die Spezifikation der durch von Komponenten verwendeten RTE-Funktionen. Somit ermöglicht SystemDesk nicht nur die Generierung von Code für einzelne Softwarekomponenten, sondern auch die Generierung Steuergeräte-spezifischer RTE-APIs sowie die Simulation der Interatktion beider Systemschichten. Die Konfiguration der notwendigen Basic Software wird hingegen nicht unterstützt, so dass zur Generierung des vollständigen Laufzeit-Stacks und der lauffähigen Applikationssoftware zusätzliche Konfigurations- und Modellierungswerkzeuge verwendet werden müssen.

In Abbildung 4.3 ist das Architekturprojekt in der Hauptansicht von SystemDesk zu sehen. Neben der graphischen bzw. tabellarischen Spezifikation der Bestandteile der einzelnen Architektursichten im Zentralbereich sind alle Architekturbestandteile eines Projektes hierarchisch im Projektbaum auf der linken Seiten strukturiert abgelegt. Im Folgenden sollen die einzelnen Architektursichten und -Entscheidungen bis hin zur Systemarchitektur kurz beschrieben werden.

#### Modellierung der Softwarearchitektur

Vor der Modellierung der eigentlichen Kompositionsstruktur für das Softwaresystem können zunächst in der *ProjectLibrary* vordefinierte *Typen* für die Bestandteile des Architekturentwurfs definiert werden, die später mehrfach bzw. wiederverwendet werden sollen. Dazu zählen neben ganzen Komponenten oder Kompositionen vor allem Standardelemente, die häufig an verschiedenen Stellen verwendet werden. Abbildung 4.4 zeigt die Definition exemplariscchen Standarddatentypen und Interface-Typen, die später mehrfach für die Spezifikation zur Typisierung von Datenelementen von Schnittstellenspezifikationen bzw. zur Typisierung von Kommunikationsports verwendet werden. Der Entwurf der Soft-

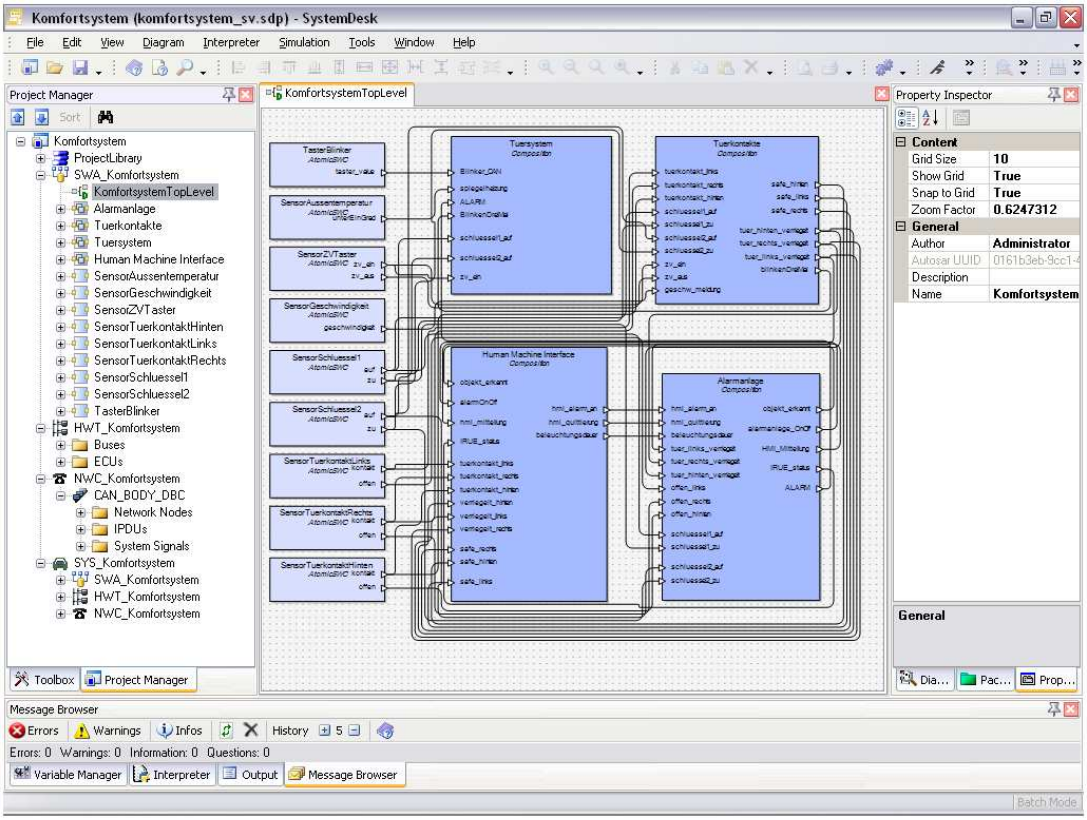


**Abbildung 4.3:** Architekturmodellierung in SystemDesk

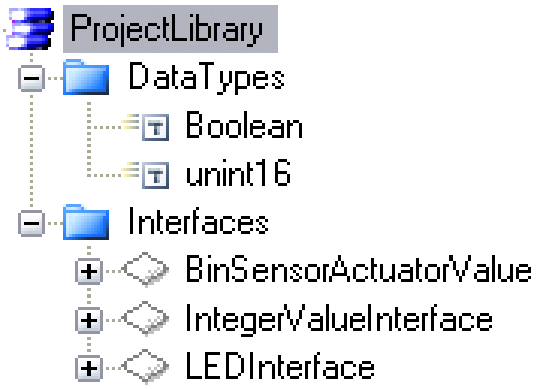


**Abbildung 4.4:** Interface und Type Library in SystemDesk

warearchitektur erfolgt in Hinblick auf die umzusetzenden funktionalen Anforderungen durch Auswahl und Kombination geeigneter Teilsysteme. Die entsprechenden Funktionsblöcke, die in SystemDesk gemäß des AUTOSAR-Standards entwe-

der atomare Komponenten oder hierarchische Kompositionen sind, werden durch Portdefinition von außen nutzbar gemacht. Die Funktionalität einer Komponente wird durch die Typisierung dieser Ports über entsprechende Interfaces vorgenommen. Spezielle Softwarekomponenten stellen dabei Sensor/Aktor-Komponenten dar, die entsprechend der Rolle dieser Art von Systemelement reine Datenquellen- oder Senken im System repräsentieren. Die Spezifikation von Wirkketten zur späteren Implementierung komplexer Funktionalität erfolgt durch Kommunikation zwischen Sensorik, Aktorik und Komponenten mit Logik zur Regelung und Steuerung der jeweiligen Abläufe.

Bei einer Top-Down-Spezifikation einer Softwarearchitektur beginnt der Entwurf mit der Definition einer Toplevel-Komposition, die logische Wurzel der Softwarefunktionen darstellt und alle weiteren Teilsysteme als Unterkomponenten enthält. Die in Abbildung 4.5 dargestellte Toplevel-Komposition für das Komfortsystem der Fallstudie enthält auf äußerster Ebene die vier großen Untersysteme HMI, Türsystem, Türkontakte und Alarmanlage. Zudem befinden sich auf dieser Ebene Sensorik-Elemente des Systems, deren Sensordaten in verschiedene Untersysteme delegiert werden, wie bespielsweise Türkontakte und Geschwindigkeitsmesser. Während die Untersysteme der Alarmanlage und der Türkontakte aus jeweils einer Komponente zur Umsetzung der entsprechenden Funktionalität bestehen, ist interne Realisierung des HMI und des Türsystems wesentlich umfangreicher. Das HMI wurde aufgrund der komplexen Ansteuerung des Display später als eine einzige atomare Software-Komponente direkt in C implementiert und wird an dieser Stelle nicht weiter betrachtet.

Nachfolgend soll kurz auf die weitere Dekomposition des Türsystem eingangen werden, die in Abbildung 4.6 dargestellt ist. Das Türsystem untergliedert sich weiter in Unterkompositionen für den Fensterheber (siehe Abbildung 4.7) und die Spiegelverstellung (Abbildungen 4.8 und 4.9), atomare Komponenten der Spiegelheizung, den Blinker und den Türöffner sowie spezielle, in diesem Teilsystem verwendete Sensorik/Aktorik. Schließlich ermöglicht SystemDesk auch die Spezifikation der internen Details atomarer Komponenten, dem *InternalBehavior*. Dazu gehören unter anderem die Definition von einem oder mehreren Prozessen zur Umsetzung der durch die Schnittstellenspezifikation beschriebenen Funktionalität der Komponente. Diese *Runnables* können entweder ereignisgesteuert oder zyklisch ausgelöst werden. Darüber hinaus können weitere interne Details wie z.B. Interprozess-Variablen definiert werden. Die eigentliche Umsetzung der *Runnables*, entweder durch entsprechende Verhaltensmodelle und anschließender Code-Generierung oder direkt in Code, erfolgt dann in der Phase des technischen Designs bzw. der Implementierung.

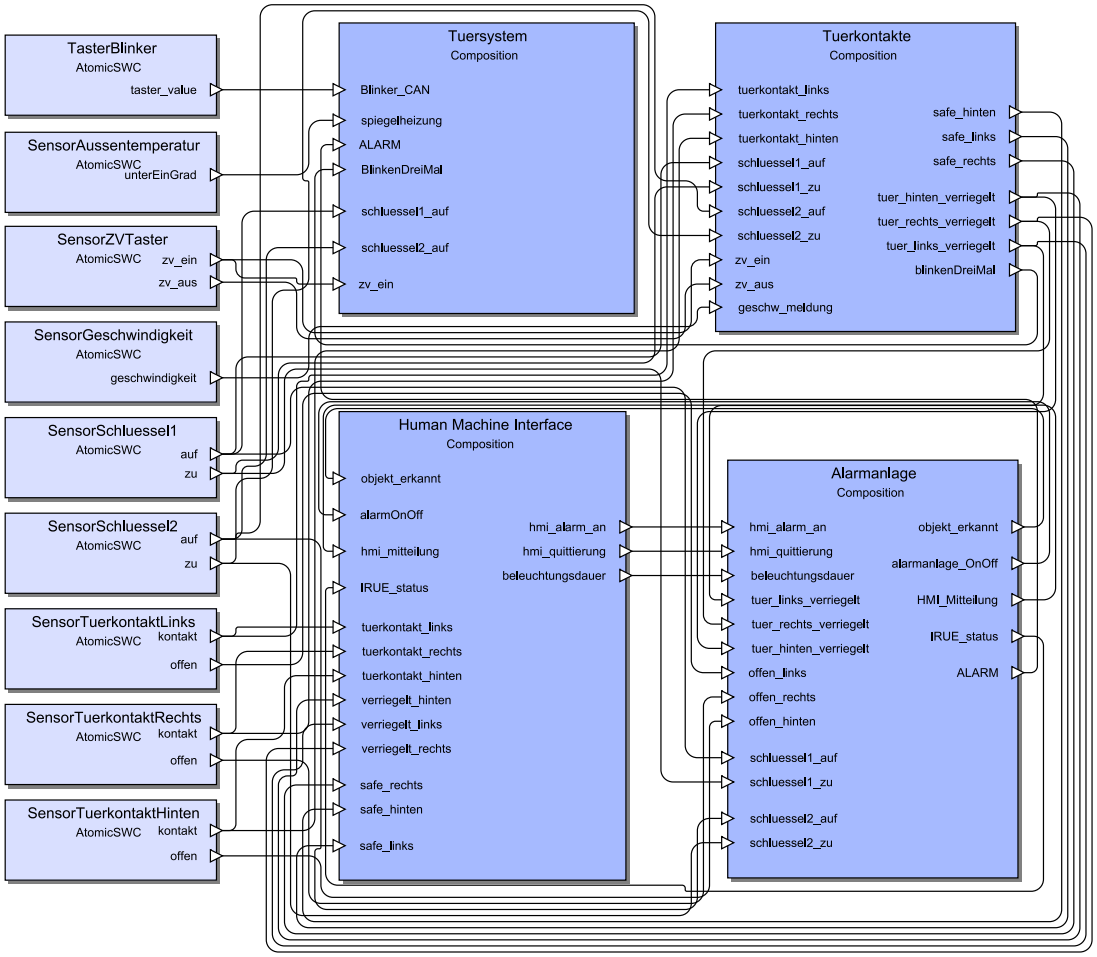


**Abbildung 4.5:** Toplevel-Komposition

### Modellierung der Hardware-Topologie

Die Modellierung der Hardware-Topolgie erfolgt in SystemDesk tabellarisch durch Definition der einzelnen, im System verbauten ECUs und deren Anbindungen an ein Bussystem. Abbildung 4.10 zeigt die Topologie-Spezifikation für das Komfortsystem aus der Fallstudie. Es wurde jeweils eine ECU für jedes der vier Hauptkomponenten des Systems, doorEcu, lockEcu, HmiEcu und AlarmEcu, vorgesehen sowie eine ECU als *„Dummy“*-Repräsentation des Simulationsrechners für die Restbussimulation nicht in Hardware realisierter Systembestandteile. Die Integration der Steuergeräte erfolgt über einen gemeinsamen CAN-Bus.

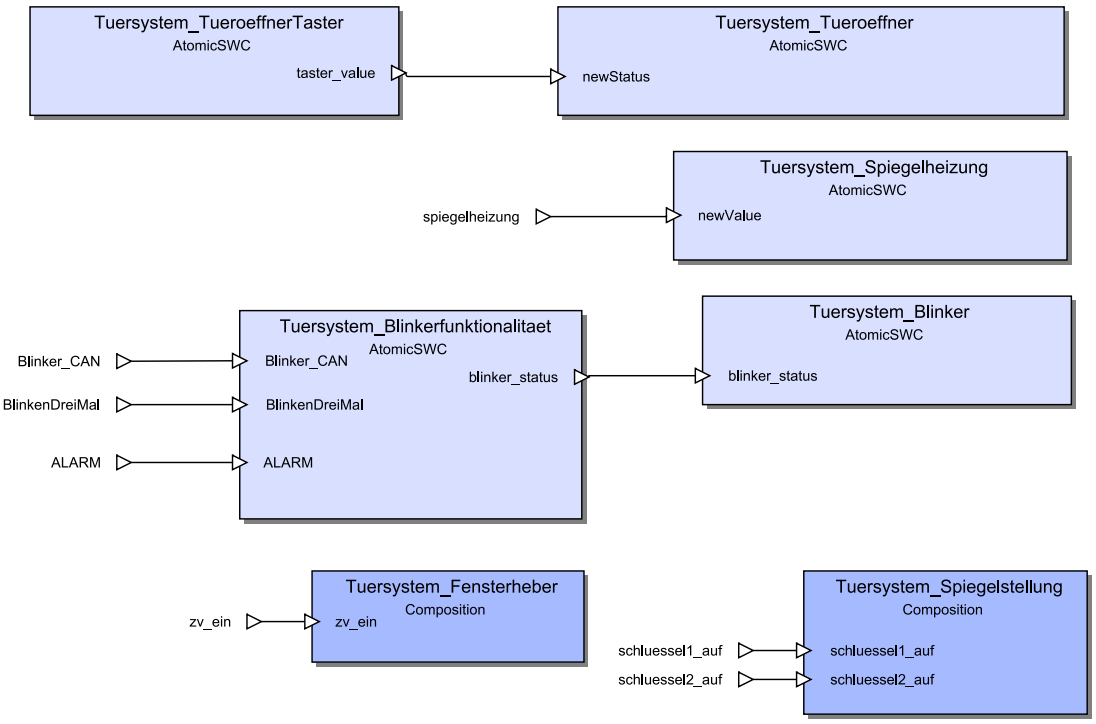


**Abbildung 4.6:** Komposition Türsystem

### Integration der Systemarchitektur

Nach der Spezifikation der Softwarearchitektur und Hardware-Topologie erfolgen schließlich die eigentliche Architektur-Entscheidungen durch die Integration beider Teilsichten in eine Systemarchitektur:

1. Verteilung der atomaren Softwarekomponenten auf Steuergeräte
2. Definition der Kommunikationsmatrix, d.h. Zuordnung von Hardware-Ports der Steuergeräte zu Netzwerkknoten
3. Definition von Bussginalen für die Kommunikation von Datenelementen zwischen Netzwerkknoten

Die Verteilung der vier Hauptkomponenten des Komfortsystems sowie die zugehörigen Sensor/Aktor-Komponenten auf die entsprechenden Steuergeräte erfolgt durch Drag & Drop im Projektbaum. Das Ergebnis des Mappings ist in Abbildung 4.11 zu sehen. Die Definition der Kommunikationsmatrix erfolgt durch die Zuordnung der Hardware-Ports der Steuergeräte aus der Hardware-Topologie zu zugehörigen Netzwerkknoten, siehe Abbildung 4.12. Nun erfolgt die Definition von Bussignalen zwischen Netzwerkknoten zur Umsetzung des Austauschs

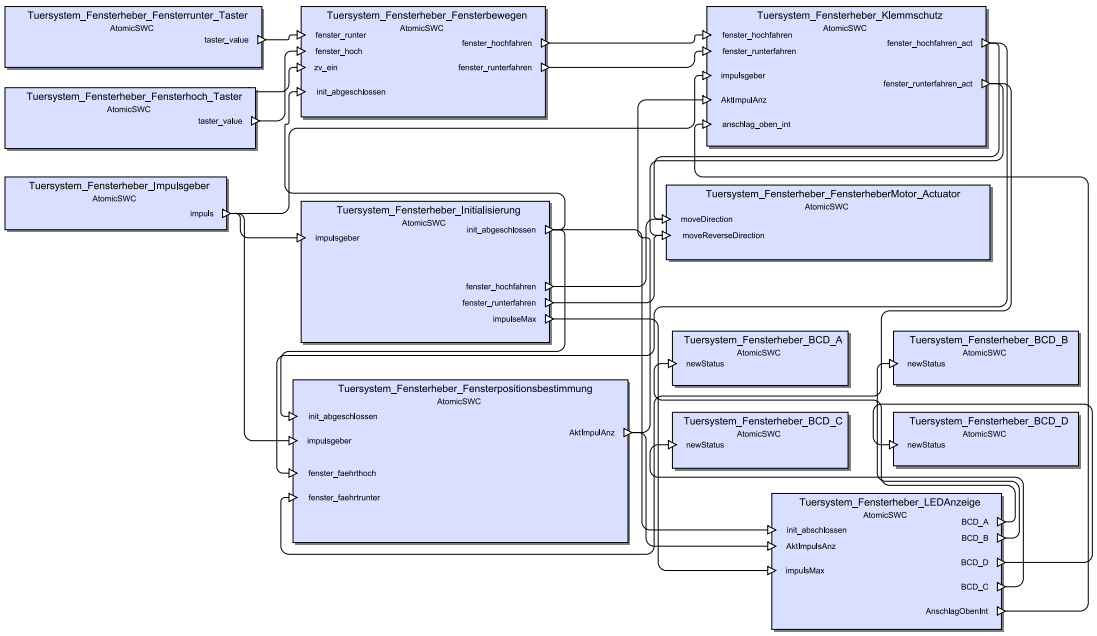


**Abbildung 4.7:** Komposition Fensterheber

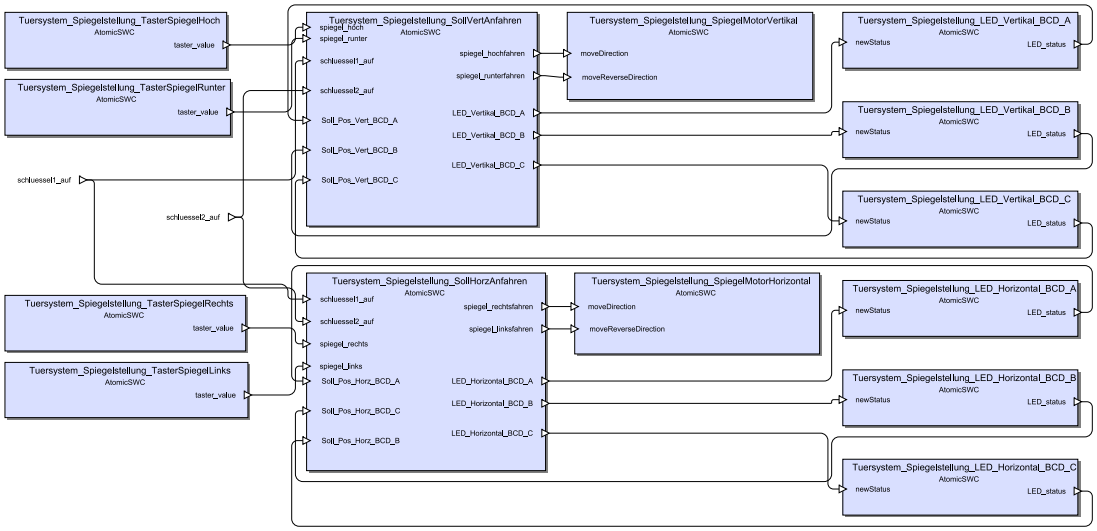


**Abbildung 4.8:** Komposition Spiegelverstellung

von Datenelementen über die Schnittstellen von zwei über einen Konnektor miteinander kommunizierenden, verteilten Softwarekomponenten. Abbildung 4.13 zeigt exemplarisch die Signaldefinition für das Türsystem der Fallstudie. Ein Ausschnitt aus dem resultierenden Projektbaum für die vollständige Spezifika-


**Abbildung 4.9:** Komponente zur Spiegelverstellung

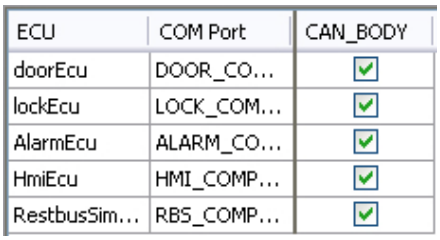


**Abbildung 4.10:** Steuergeräte-Topologie in SystemDesk

tion der Systemarchitektur in SystemDesk ist in Abbildung 4.14 zu sehen. Für die nachfolgenden Phasen der AUTOSAR-Methodik, also der Verhaltesmodellierung der Softwarekomponenten sowie der System-Konfiguration, ermöglicht SystemDesk den Export der Architektur-Spezifikation im durch den Standard vorgegebenen AUTOSAR XML-Format (siehe Abbildug 4.15). Entsprechende Authoring Tools zur Generierung von Applikationscode und RTE können diese Architektur-Beschreibung als grundlegende Systemstruktur importieren und dann um entsprechende Implementierungsdetails verfeinern verfeinern.

## 4.3.2 Architekturevaluation

### Motivation

Neben der Menge funktionaler Anforderungen, auf deren Grundlage die Spezifikation der Systemarchitektur vorgenommen wurde, sind weitere nicht-funktionale bzw. extra-funktionale Anforderungen durch das System zu erfüllen. Diese Anforderungen beziehen sich in der Regel nicht auf ein bestimmtes Feature, sondern beschreiben allgemeine, übergeordnete Eigenschaften, die das Gesamtsystem betreffen.

Die zunehmende Bedeutung, die Qualitätskriterien für Entwurfsentscheidungen neuer Systeme haben, wird exemplarisch nicht nur an dem Kriterium Kosten deutlich, das den treibenden Faktor, insbesondere im Bereich der Massenproduktion darstellt. Vielmehr spielen vermehrt auch die Einhaltung von Sicher-

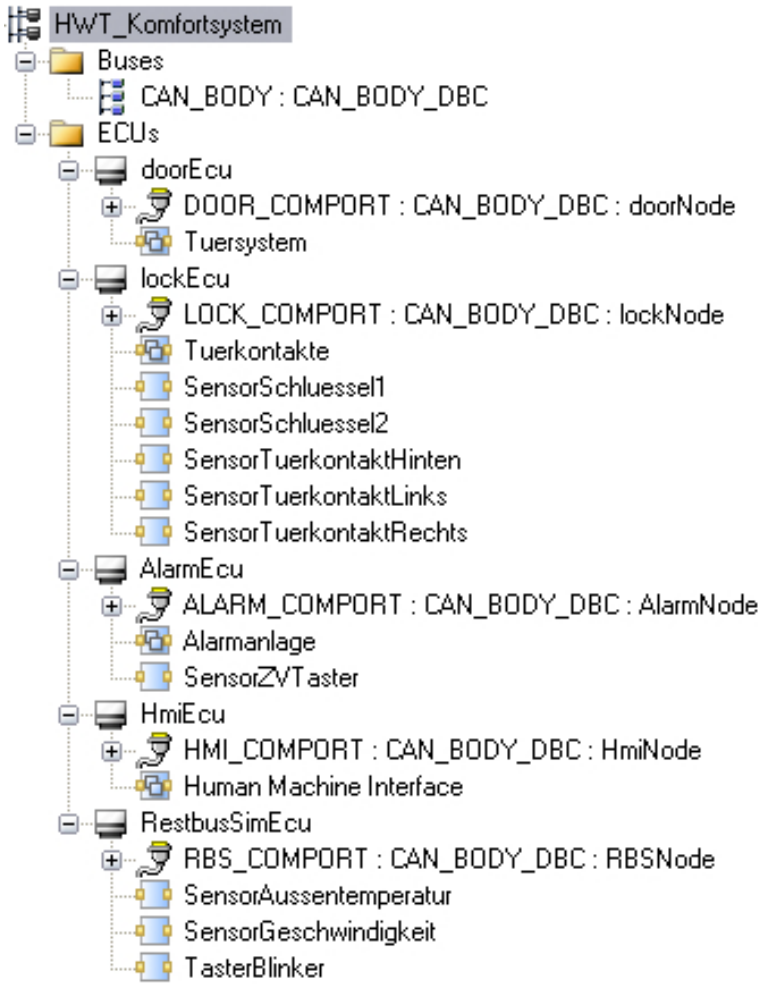


**Abbildung 4.11:** Software Component Mapping

| Bus | Communication Matrix | Communication Port | Node |
|---|---|---|---|
| CAN_BODY | CAN_BODY_DBC | DOOR_COMPORT (doorEcu) | doorNode |
| | | LOCK_COMPORT (lockEcu) | lockNode |
| | | ALARM_COMPORT (AlarmEcu) | AlarmNode |
| | | HMI_COMPORT (HmiEcu) | HmiNode |
| | | RBS_COMPORT (RestbusSimEcu) | RBSNode |

**Abbildung 4.12:** Network Mapping

heitsanforderungen, nicht zuletzt vonseiten des Gesetzgebers wie z.B. durch SIL (Sicherheits-Integritätslevel, IEC 61508/IEC61511 gefordert, bereits im Rahmen des Architekturentwurfs eine entscheidende Rolle für die Qualität eines Systems. Entsprechende Standardisierungsmaßnahmen (z.B. ISO/IEC 9126) für alle maß-

Ecu: doorEcu

| Component | Port | Interface | Data Element | Direction | Target component | Target ECU | Bus | IPDU | System Signal |
|---|---|---|---|---|---|---|---|---|---|
| | Blink... | BinSensor... | value | | TasterBlinker | RestbusSimEcu | CA... | Blink... | Blinker_CAN |
| Tuersystem_Blin... | Blink... | BinSensor... | value | | Tuerkontakte_Tuerve... | lockEcu | CA... | Blink... | BlinkenDreiMal |
| | ALARM | BinSensor... | value | | Alarmanlage_Alarman... | AlarmEcu | CA... | Alarm | Alarm |
| Tuersystem_Fe... | zv_ein | BinSensor... | value | | SensorZVTaster | AlarmEcu | CA... | Zent... | Zentralverriegel... |
| Tuersystem_Spi... | new... | BinSensor... | value | Rx | SensorAussentemper... | RestbusSimEcu | CA... | Auss... | Aussentemperatur |
| Tuersystem_Spi... | schlu... | BinSensor... | value | | SensorSchluessel1 | lockEcu | CA... | Schlu... | Schluessel1_auf |
| | schlu... | BinSensor... | value | | SensorSchluessel2 | lockEcu | CA... | Schlu... | Schluessel2_auf |
| Tuersystem_Spi... | schlu... | BinSensor... | value | | SensorSchluessel1 | lockEcu | CA... | Schlu... | Schluessel1_auf |
| | schlu... | BinSensor... | value | | SensorSchluessel2 | lockEcu | CA... | Schlu... | Schluessel2_auf |

**Abbildung 4.13:** Signal Mapping

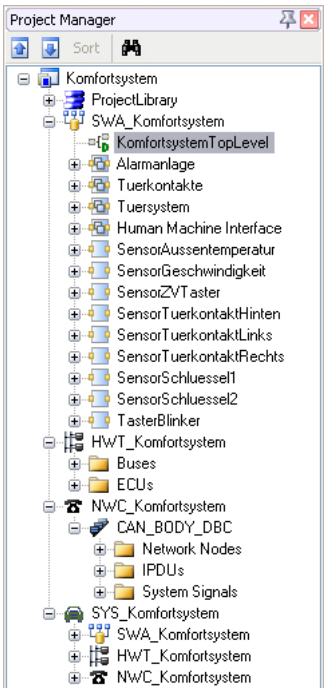


**Abbildung 4.14:** Architekturprojekt in SystemDesk

geblichen Qualitätskriterien, insbesondere für Software-Systeme, machen ebenfalls die wachsende Bedeutung dieser Faktoren deutlich. Neben der Schwierigkeit, diese Art von Kriterien mit Bezug auf ein konkretes System geeignet zu charakterisieren, besteht die grundlegende Problematik dann vor allem darin, dass etablierte Techniken zur Sicherung der Systemqualität auf funktionaler Seite (Fehlerfreiheit), wie beispielsweise Testen oder Simulation, hierfür in der Regel nicht adäquat sind. In allen anderen Fällen würden mithilfe dieser Techniken erkannte Fehler, wie zum Beispiel Mängel in der Verfügbarkeit einer bestimmten Funktionalität erst sehr spät im Entwicklungsprozess erkannt werden können.

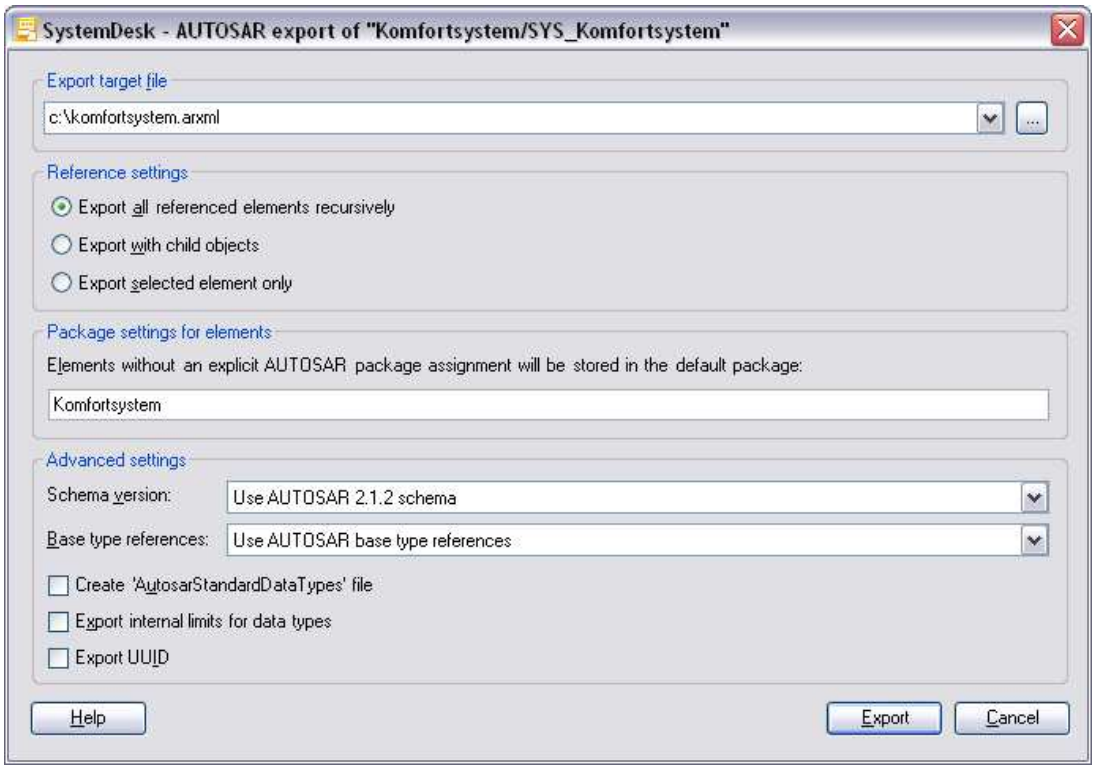


**Abbildung 4.15:** AUTOSAR-Export aus SystemDesk

Da die notwendigen Korrekturen durch die große Anzahl betroffener Systemartefakte mit einem enorm hohen Aufwand verbunden wären, würde dies zu hohem Zusatzaufwand und entsprechenden Kosten führen.

Eine weitere Herausforderung bei der Einhaltung projektspezifischer Qualitätsanforderungen besteht darin, dass die verschiedenen Optimierungsziele in der Regel gegenläufig sind, also ein geeigneter Kompromiss gefunden werden muss. Darüber hinaus muss eine Einschätzung der zu erwartenden Gesamtqualität des Systems möglichst frühzeitig begleitend zum Systementwurf erfolgen, um eventuell notwendige Optimierungen nicht zu aufwendig und kostspielig werden zu lassen.

Der Architektur-Entwurf stellt eine hierfür geeignete Spezifikationsphase dar. Der Entwurf einer Systemarchitektur stellt die früheste Design-Entscheidung für ein neu zu entwickelndes System dar und legt die Grobstruktur des Gesamtsystems zur Umsetzung der geforderten Funktionalität fest. Wie zuvor eingeführt, ist das wesentliche Modellierungskonzept miteinander interagierende Komponenten, die Grundelemente sowohl des Softwaresystems in Form von Funktionsblöcken, als auch des Hardwaresystems (z.B. Steuergeräte, Busse, Sensorik/Aktorik) repräsentieren. Die von einer Komponente im System gelieferte Teilfunktionalität

wird durch invariante Schnittstellenspezifikationen beschrieben, von der konkreten Implementierung bzw. technischen Realisierung wird hingegen zunächst abstrahiert. Ein weiterer Abstraktionsansatz im Rahmen des Architektur-Entwurfs stellt das Sichtenkonzept dar, also der zunächst getrennte Entwurf der Software-Architektur und Hardware-Topologie des Steuergerätenetzes und deren abschließende Integration in eine Systemarchitektur. Diese Integration, die die eigentliche Architektur-Entscheidung darstellt, erfolgt durch die Zuordnung (Mapping) der Softwarekomponenten auf die Steuergeräte, auf denen sie ausgeführt werden sollen sowie durch die Abbildung von zwischen Komponenten verschiedener Steuergeräte ausgetauschten Datenelementen auf Bussignale. Die konsequente Dekomposition und Hierarchisierung des Systems kann wesentlich zur Beherrschbarkeit der steigenden Systemkomplexität beitragen und zudem die einfache Wiederverwendbarkeit und Austauschbarkeit von Teilsystemen ermöglichen. Somit erlauben diese Konzepte eine beliebige, funktionsorientierte Modellierung von Architekturvarianten sowie flexible Anpassungen der Grundbestandteile des Systems. Architekturspezifikationen beinhalten zugleich aber bereits hinreichend detaillierte Entwurfsentscheidungen, um aussagekräftige Abschätzungen über die zu erwartende Qualität des implementierten Systems durch Architektur-Evaluation geben zu können.

Die Architektur-Evaluation stellt also eine viel versprechende Technik dar, um frühzeitig eine Abschätzung der zur erwartenden Qualität des Systems bereits auf Grundlage dessen Grobstruktur unabhängig von der späteren Implementierung vornehmen zu können. Durch eine Kombination geeigneter Bewertungstechniken wie Metriken, Lastprofile und Simulationsergebnisse sowie Expertenbefragungen können so hinreichend aussagekräftig verschiedene Architekturvarianten zur Umsetzung der Gesamtfunktionalität hinsichtlich ihrer Qualität bewertet und verglichen werden. Verschiedene, existierende Ansätze zur Architekturevaluation können durch die Integration in einen systematischen Entwicklungsprozess zur Optimierung der Kosten in Fahrzeugprojekten beitragen und die Festlegung eines geeigneten Tradeoffs zwischen Kosten und Qualität des Systems unterstützen. Eine systematische Evaluationsstruktur kann zugleich als Dokumentation von Entwurfsentscheidungen dienen.

Obwohl letztendlich die Qualität durch Entscheidungen in allen Entwicklungsphasen beeinflusst werden, kann die besondere Bedeutung der Architektur durch folgende Aussagen charakterisiert werden:

- Eine *gute* Systemarchitektur ist die Grundvoraussetzung für die Erfüllung von Qualitätskriterien durch ein System.

- Eine *schlechte* Systemarchitektur kann die Erfüllung von Qualitätskriterien durch ein System von vornherein unmöglich machen.

Eine gewählte Architekturvariante kann demnach nicht per se als *„gut“* oder *„schlecht“* bezeichnet werden, sondern nur in Bezug auf die vom System geforderten Qualitätskriterien und dem Kontext (Szenarien, Umfeld, Randbedingungen, etc.), in dem es eingesetzt werden soll. Verschiedene Varianten für eine mögliche Architektur können sich durch unterschiedliche Architekturentscheidungen in den drei beschriebenen Ebenen ergeben, wobei als Ausgangspunkt stets eine invariante Systemfunktionalität angenommen wird. Je nach Auswahl und Gewichtung der zu betrachtenden Kriterien für eine Architekturevaluation kann das Ergebnis dann sehr unterschiedlich ausfallen.

### Allgemeine Vorgehensweise

Die Auswahl, Gewichtung und das geforderte Maß der durch ein System zu erfüllenden Qualitätskriterien können durch zahlreiche Einflüsse bestimmt werden, wie z.B. durch Forderungen der Stakeholder oder zu erfüllende Normen und Standards. Dabei sind Qualitätsattribute in der Regel nicht für ein bestimmtes System an sich bestimmbar, sondern müssen anhand von allgemeinen systemunabhängigen oder systemspezifischen Anwendungsszenarien für dieses konkrete System beschrieben werden.

Die wesentlichen Bestandteile eines *Szenarios*, mit dem ein Qualitätsattribut charakterisiert wird besteht aus folgende Faktoren [BCK03]:

1. Stimulus: Eine Konstellation, deren Auftreten das System beeinflusst, wie z.B. ein Signal, das zu einem internen Zustandswechsel führt oder die Berechnung einer Antwort für eine Benutzeranfrage.

2. Quelle des Stimulus: Abstrakte oder konkrete Einheit in oder außerhalb des Systems, wie z.B. der Benutzer oder ein Sensor.

3. Artefakt: (Teil-) System, dass durch den Stimulus angeregt wird, wie z.B. eine bestimmte Prozedur zur Berechnung einer Antwort.

4. Umgebung: Bedingungen (intern und extern), unter denen der Stimulus auftritt und zu bearbeiten ist, wie z.B. die Prozessorauslastung

5. Maßeinheit für die Bewertung der Antwort (Response) auf den Stimulus, also dem Qualitätsattribut. Beispielsweise kann die Performanz anhand der Antwortzeit auf eine Anfrage bestimmt werden.

Neben den oben genannten, intuitiven Beispielen zeigt Abbildung 4.16 die Anwendung dieses Schemas auf das Qualitätsattribut *„Modifizierbarkeit“*. Die Auswertung der zu betrachtenden Qualitätsattribute für eine Architekturvariante gemäß dem zugehörigen Maß in entsprechenden Szenarien erfolgt mit Hilfe von *Bewertungstechniken*, wobei im Allgemeinen zwischen zwei methodischen Ansätzen unterschieden wird:

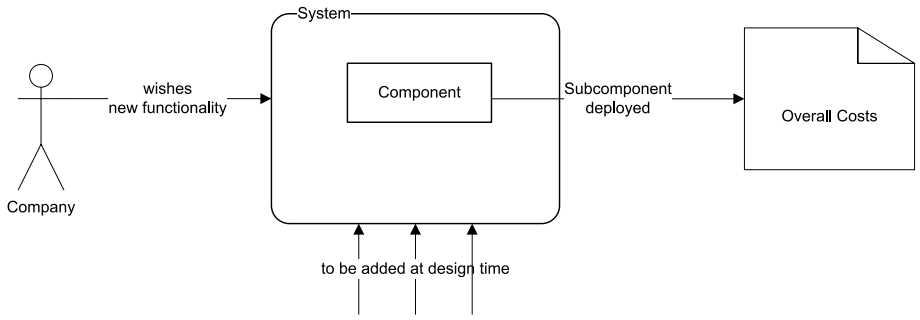


**Abbildung 4.16:** Szenario für Modifizierbarkeit

1. **Quantitative Methoden**, wie z.B. Messverfahren, Metriken und Simulationsergebnisse.
2. **Qualitative Methoden**, wie z.B. Befragung von Experten und Erfahrungswerte.

Neben der oben genannten ISO-Norm gibt es zahlreiche Ansätze zur Einordnung verschiedener Qualitätskriterien. Beispielsweise können folgende, grundlegende Kategorien von Bewertungskriterien unterschieden werden, in denen dann beliebig spezialisierte Unterkriterien betrachtet werden:

1. **Performanz**
   - Antwortzeiten für Funktionen (Prozessorauslastung)
   - Benötigte Datenmengen von Funktionen (Speicherauslastung)
   - Kommunikationsaufkommen zwischen Funktionen auf verschiedenen Steuergeräten (Buslast)
2. **Kosten**
   - Hardware (Summe der Bauteilkosten und Verkabelung)
   - Software (Entwicklungskosten des Softwaresystems oder Zukauf von Zulieferern)
   - Gewicht
3. **Modifizierbarkeit**
   - Erweiterbarkeit (Hinzufügen weiterer, neuer Funktionalität)
   - Skalierbarkeit (Vermehrung oder Verringerung vorhandener Funktionalität)
   - Spezialform: Portierbarkeit (auf eine andere Plattform)

Daneben gibt es weitere, allgemeiner gefasste Kriterien, wie z.B.:

- Verfügbarkeit (z.B. Ausfallwahrscheinlichkeit)
- Sicherheit (z.B. benötigter Aufwand, um unbefugt auf Daten zuzugreifen)
- Testbarkeit (z.B. Wahrscheinlichkeit, dass ein auftretender Fehler erkannt wird)
- Nutzbarkeit (z.B. mittlere Antwortzeit oder mittlere Anzahl auftretender Fehler)

Nach der Bewertung der Qualitätsattribute, d.h. der Datenermittlung und Anwendung der Bewertungsfunktion, kann im nächsten Schritt die Integration der Bewertungsergebnisse in einem einheitlichen Evaluationsverfahren zur Ermittlung einer Gesamtqualität erfolgen. Hierzu sind zunächst geeignete Qualitätsraten zur Normalisierung der verschiedenen Kriterien zu definieren, um die Vergleichbarkeit der unterschiedlichen Kriterien zu ermöglichen (siehe nachfolgend).

Nachfolgend sollen kurz einige Ansätze zum Erreichen von Qualitätszielen skizziert werden. Das Erreichen von geforderten Qualitätszielen für ein System kann durch Design-Entscheidungen im Rahmen der Systemarchitektur maßgeblich beeinflusst werden. Für die verschiedenen Kriterien gibt es jeweils entsprechende *Taktiken*, also Architekturrichtlinien bzw. vordefinierte Design-Entscheidungen, deren Anwendung zum Erreichen von Qualitätszielen beitragen können.

Die Menge von Taktiken, die zur Erfüllung von Qualitätszielen durch die Entwickler herangezogen werden, definiert die *Strategie*, die bei der Definition der Systemarchitektur angewendet wurde. Eine bestimmte Strategie für die Umsetzung einer allgemeinen Aufgabenstellung in einem System ist dann durch ein Architektur-Pattern vorgegeben. Weitere Details und Beispiele können [BCK03] entnommen werden.

Die Gesamtqualität eines Systems und dessen Architektur, die gemäß der im vorherigen Abschnitt beschriebenen Vorgehensweise definiert wurde, ergibt sich aus einer Vielzahl von Einflüssen, die im folgenden erläutert werden sollen. Das beschriebene Verfahren basiert auf dem ganzheitlichen Evaluationsansatz zur Integration verschiedenster Kriterien und Bewertungstechniken, wie in [FGB+07] vorgeschlagen wurde.

Zunächst müssen die für die Evaluation relevanten Qualitätsattribute bestimmt werden. Für die so definierten Qualitätskriterien können nun die Zielsetzungen für die Optimierung festgelegt werden, wie z.B.:

- Die Gesamtkosten des Systems sollen möglichst gering sein.
- Die Antwortzeit für einen bestimmten Dienst soll möglichst gering sein.
- Die maximale Buslast (Überlast) darf nicht überschritten werden.
- Die Verfügbarkeit soll möglichst hoch sein.

Diese Zielsetzungen können dann zur Evaluation entsprechend präzisiert werden, z.B. durch die Angabe von K.O.-Kriterien, also Wertegrenzen, die für ein brauchbares System nicht überschritten werden dürfen (z.B. ein finanzielles Gesamtbudget).

Insbesondere *„unschärfere“* Kriterien wie z.B. Modifizierbarkeit können zudem durch entsprechende Anforderungsszenarien präzisiert werden:

- Skalierbarkeit: Die Türsteuerungssysteme für einen 3-Türer sollen auch in einem 5-Türer einsetzbar werden.

- Erweiterbarkeit: Das Feature *„Einklemmschutz“* des Fensterhebers soll in Fahrzeugen für den europäischen Markt vorhanden sein, für den amerikanischen Markt hingegen nicht.

Im nächsten Schritt müssen für diese Kriterien entsprechende Maßzahlen (Metriken) definiert werden, um die Attribute auf Zahlenwerte abbilden zu können. Das können entweder Messwerte (z.B. Antwortzeit) bzw. errechnete Werte (z.B. Kosten, Verfügbarkeit, Anzahl zusätzlich ausführbarer Prozesse auf einem Steuergerät) oder qualitative Werte (z.B. Ranking durch Befragung) sein.

Die konkreten Ergebnisse für ein betrachtetes System ergeben sich dann durch Anwendung der zugehörigen Bewertungstechniken (siehe oben).

Im letzten Schritt erfolgt die Einordnung von Einzelergebnissen für Qualitätsattribute in dessen Gesamtwerteraum. Hierzu bildet die Qualitätsrate das Bewertungsergebnis für ein bestimmtes Attribut auf dessen *„Güte“* in Prozent ab, wobei 0 Prozent die geringste Qualität und 100 Prozent das Optimum repräsentiert. Die Qualitätsrate stellt demnach das Evaluationsergebnis für ein einzelnes Attribut des Systems dar. Der Verlauf einer solchen Kurve spiegelt die Auswertung von Änderungen eines Qualitätsattributes wieder.

Abbildung 4.17 zeigt einen exemplarischen Verlauf der Qualitätsrate für die Buslast. Bis zu einem bestimmten Punkt im Bereich geringer Lasten wird eine konstante Rate von 100 Prozent angenommen, d.h. eine unerhebliche Auslastung des Bussystems. Auf der anderen Seite erreicht die Rate ab einem bestimmten Wert, der in der Regel schon unterhalb der hundertprozentigen Auslastung liegt, einen Wert von 0 Prozent, wird also zum K.O.-Kriterium und das System ist nicht umsetzbar.

Der Bereich dazwischen verläuft in der Regel nicht linear, so dass Änderungen an Qualitätskriterien je nach Gradient im betrachteten Punkt unterschiedlich starke Auswirkungen auf deren Evaluation haben können. Der Verlauf für eine Qualitätsrate entsteht in der Regel aus Expertenwissen.

Abschließend werden die verschiedenen (in der Regel heterogenen) Kriterien und deren entsprechenden Evaluationsergebnisse gemäß ihrer Auswertungen und zugehörigen Raten in ein Gesamtevaluationsergebnis für die betrachtete Architek-

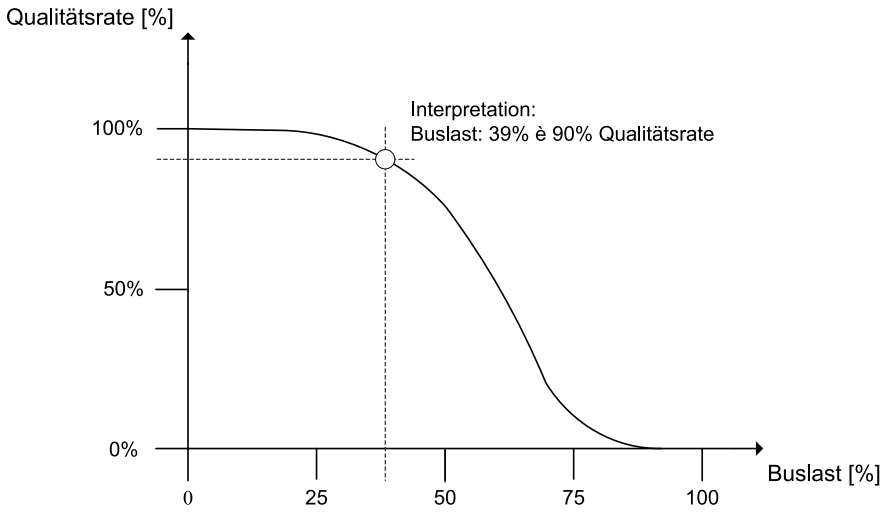


**Abbildung 4.17:** Qualitätsrate für die Buslast

turvariante integriert. Bei der Strukturierung der Einzelkriterien sollen folgende Aspekte berücksichtigt werden:

- Hierarchie: Zu allgemeinen Kriterien (z.B. Kosten) sollen Unterkriterien (z.B. Hardware- und Softwarekosten) angegeben werden, deren Einzelauswertung zusammen die Bewertung des Oberkriteriums ergeben. Dabei sind die Hierarchie-Ebenen beliebig schachtelbar, d.h. Unterkriterien können weitere Unterkriterien enthalten, etc.

- Gewichtung: Bei der Zusammensetzung von Oberkriterien aus mehreren Unterkriterien sollen diese unterschiedlich gewichtet werden. Beispielsweise könnten Kosten eine höhere Priorität und somit ein höheres Gewicht bei der Gesamtevaluation haben, als die Modifizierbarkeit. Diese Prioritäten werden in Allgemeinen in Absprache mit den Stakeholdern festgelegt. Hierbei wird noch einmal deutlich, dass verschiedene Qualitätsattribute durchaus gegenläufig sein können, so dass eine *„optimal“* bewertete Architektur immer im Zusammenhang mit der Gewichtung der Einzelattribute zu sehen ist.

Zur Darstellung dieser Zusammenhänge bietet sich eine Graphenstruktur an, wie sie in Abbildung 4.18 [FGB+07] dargestellt ist. Die äußeren Knoten des Graphen (Blätter) beschreiben jeweils ein konkretes Qualitätskriterium, das bewertbar ist und dem eine Qualitätsrate zugewiesen werden kann. Die inneren Knoten dagegen stellen zusammengesetzte Attribute dar, deren Auswertung sich aus der Kombination der direkten Unterkriterien (Unterknoten) ergibt. Je nach Priorität der Unterkriterien gehen die Unterkriterien mit unterschiedlicher Gewichtung in das Gesamtergebnis ein.

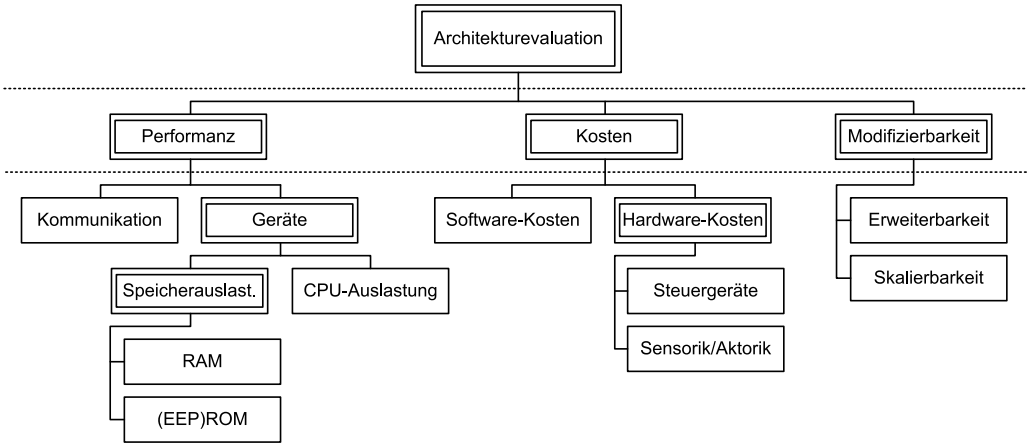


**Abbildung 4.18:** Evaluationsstruktur

Die Auswertung von zusammengesetzten Kriterien soll anhand des Graphenausschnitts in Abbildung 4.19 erläutert werden. Die drei Unterkriterien sind je-

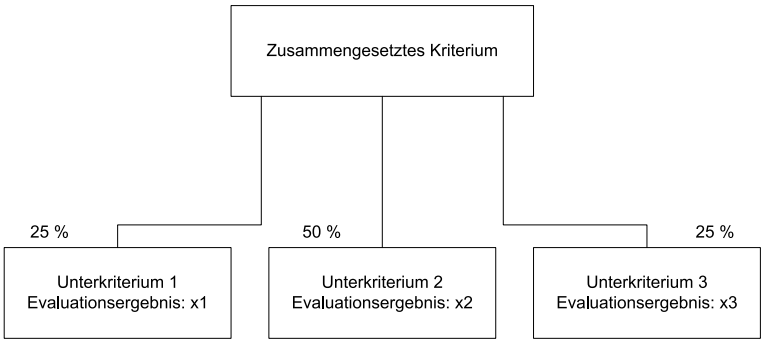


**Abbildung 4.19:** Zusammengesetztes Qualitätskriterium

weils mit 25 Prozent, 50 Prozent und 25 Prozent gewichtet. Folglich ergibt sich für die (normierte) Evaluation des zusammengesetzten Kriteriums:

$$(0.25xx1) + (0.5xx2) + (0.25xx3)$$

Die Berechnung der Gesamtqualität eines Systems erfolgt also durch bottom-up Berechnung der zusammengesetzten Attribute, angefangen bei den Blättern, also den konkreten Kriterien, bis hin zur Wurzel, die alle Teilergebnisse integriert. Demnach ist durch die Anordnung des Graphen die Reihenfolge der Evaluation vorgegeben. Die Informationen, die zur Durchführung einem Knoten zuzuordnen sind, bestehen aus den folgenden Angaben:

- Name des Kriteriums
- Bewertungsergebnis (Metrik)

- Qualitätsrate für das Bewertungsergebnis (Prozent)
- Gewichtung (Prozent)

Wird während der Evaluation ein Attribut zu 0 Prozent bzw. zu einem K.O. Kriterium ausgewertet, kann der Ablauf gestoppt werden, da die betrachtete Architekturvariante keine funktionsfähige Umsetzung des Systems erlaubt.

Der Graph in Abbildung 4.18 veranschaulicht, wie die zuvor exemplarisch genannten Kriterien in eine Gesamtstruktur integriert werden können. Die doppelt umrahmten Knoten stellen zusammengesetzte Kriterien dar, die sich aus ihren Unterkriterien berechnen.

Auf Grundlage einer Evaluationsstruktur und deren Anwendung auf Qualitätsattribute können weitergehende Untersuchungen bezüglich der Einflüsse der Kriterien untereinander, der Auswirkung von Veränderungen von Einzelattributen sowie deren Gewichtung angestellt werden. Für eine genauere Beschreibung verweisen wir auf [FGB+07].

### 4.3.3 Adaption auf AUTOSAR-Systeme

#### Konstruktive und analytische Qualitätssicherung von AUTOSAR-Systemen

Im Folgenden gehen wir davon aus, dass beim Entwurf einer Systemarchitektur die Menge funktionaler Anforderungen invariant bleibt und verschiedene Architekturvarianten zur Umsetzung dieser Anforderungen untersucht werden sollen. Die Eignung dieser Varianten wird dann in Hinblick auf deren Auswirkungen auf die zu erwartende Qualität des Systems bewertet. Zwar kann ein guter Architekturentwurf die Erfüllung der gestellten Qualitätsanforderungen nicht garantieren, da es sich dabei lediglich um eine Grobstrukturierung ohne konkrete Implementierungsdetails handelt. Dennoch ist eine frühzeitige Abschätzung der zu erwartenden Qualität unverzichtbar da andererseits ein ungeeigneter Architekturentwurf zum Scheitern ganzer Fahrzeugprojekte führen kann. Auch vor dem Hintergrund der hohen Komplexität aktueller Steuergerätnetzwerke ist die Evaluation von Qualitätskriterien bereits auf Grundlage von Architekturentwürfen trotz der *„unscharfen“* Ergebnisse auch für den Entwurf von AUTOSAR-Architekturen zukünftig ein unverzichtbarer Schritt.

Im Folgenden soll die grundsätzliche Anwendbarkeit von zuvor beschriebnenen Techniken zur Architektur-Evaluation auf AUTOSAR-konform entwickelte Systeme diskutiert werden. Hierfür beziehen wir uns exemplarisch auf die zuvor beschriebene Methodik aus [FGB+07]. Wir beschreiben die Möglichkeiten der Adaption der beschriebenen Techniken auf Architekturen von AUTOSAR-Systemen. Zur Illustration beziehen wir uns auf die Fallstudie Komfortsystem.

Durch Anwendung geeigneter Metriken, Lastprofile, Expertenwissen und weitere statistischer Verfahren können bereits in dieser Phase Aussagen über zu erwar-

tende Eigenschaften des fertigen Systems, wie zum Beispiel Kosten und Buslast, abgeschätzt werden. Zwar kann auf diese Weise die Einhaltung von Qualitätskriterien durch die spätere Implementierung nicht garantiert werden, aber die Gesamtqualitäten verschiedener Architekturvarianten können miteinander verglichen und dabei unbrauchbare Varianten frühzeitig ausgeschlossen werden.

In Hinblick auf die vorhergehend beschriebenen Herausforderungen bezüglich der zukünftig zu erfüllenden Qualitätsanforderungen verfolgt der AUTOSAR-Standard eine Reihe innovativer Konzepte, wie zum Beispiel:

- Funktionsorientierte Entwurfsmethodik zur flexiblen Strukturierung und Kombination der Systembestandteile
- Transparente Schichtenarchitektur und Abstraktionssichten zur strikten Trennung der Applikationsentwicklung von der Ablauf- und Kommunikations-Plattform
- Ansätze zur konstruktiven Qualitätssicherung, z.B. Portierbarkeit, Modifizierbarkeit, Wartbarkeit etc.

Für die korrekte Umsetzung der funktionalen Anforderungen definiert der AUTOSAR-Standard eine durchgängige, strukturierte Methodik, die einen Werkzeuggestützten, einheitlichen Entwurf bis zur Implementierung auf dem Steuergerätenetzwerk vorsieht. Modellbasierte Ansätze ermöglichen den Einsatz bewährter, auf UML/SysML basierender Techniken sowie die automatische Code-Generierung aus Verhaltensmodellen. Die konsequent verfolgten Konzepte zur Dekomposition, Separation of Concerns und Component Reuse tragen zur Beherrschbarkeit der Komplexität des Gesamtsystems in allen Phasen der Entwicklung bei.

Die eingangs beschriebenen Konzepte des Architektur-Entwurfs stellen auch das Kernkonzept der AUTOSAR-Methodik bei der Entwicklung von Steuergeräte-Software dar und ermöglichen somit die frühzeitige Integration Architekturbasierter Qualitätssicherungsansätze für nicht-funktionale Anforderungen. Das Schichtenmodell mit durchgängig standardisierten Schnittstellen auf allen Systemebenen ermöglicht die Separierung des Applikationsentwurfs von der darunter liegenden Systemschicht. Dem Systemarchitekten bleibt somit viel Spielraum für einen flexiblen, funktionsorientierten Entwurf einer geeigneten Architekturvariante zur Umsetzung der Gesamtfunktionalität. Die so erzielte Modularität, Anpassbarkeit und Adaptivität von AUTOSAR-Systemen führt direkt zu einer konstruktiven Sicherstellung entsprechender Qualitätsziele wie Modifizierbarkeit, Skalierbarkeit, Erweiterbarkeit, Portabilität sowie Wartbarkeit im gesamten Produktzyklus.

Die einfache Wiederverwendbarkeit AUTOSAR-konformer Systemkomponenten kann nicht nur zur Erhöhung der Zuverlässigkeit beitragen, sondern auch

dabei helfen, Entwicklungskosten zu reduzieren. Ebenso kann eine Verbesserung von Performanzeigenschaften durch die Verwendung von Generatoren für Steuergeräte-spezifisch optimierten, effizienten Applikations- und RTE-Code erzielt werden. Die im AUTOSAR-OS integrierten Konzepte des OSEK-Standards ermöglichen zudem die Umsetzung echtzeitkritischer und sicherheitskritischer Anforderungen. Neben der konstruktiven Unterstützung derartiger, spezifischer Qualitätsziele, sollte aber auch ein allgemeiner, analytischer Ansatz zur Bewertung beliebiger Kriterien auf Grundlage des Architekturmodells in die AUTOSAR-Methodik integriert werden.

Für die zukünftige Entwicklung von Steuergerätenetzwerken stellt sich die Frage, inwieweit derartige Ansätze zur Architekturevaluation auf AUTOSAR-konform entwickelte Systeme anwendbar sind. Im Kontext der AUTOSAR-Methodik zur Entwicklung von AUTOSAR-Systemen spielt der Architekturentwurf, insbesondere die Softwarearchitektur, eine zentrale Rolle. Daher stellt eine Adaption von Evaluationsansätzen sowie deren Integration in den AUTOSAR-Entwicklungsprozess einen viel versprechenden Schritt hin zur Kosten- und Qualitätsoptimierung von AUTOSAR-Systemen dar.

Die wesentliche Herausforderung bei der Spezifikation einer geeigneten Systemarchitektur eines AUTOSAR-Systems besteht darin, für die umzusetzende Funktionalität passende Komponenten auszuwählen und hierfür einen geeigneten Grad der Dekomposition in Teilsysteme zu bestimmen. Oftmals wird nach dieser Auswahl das Mapping der Komponenten und Datensignale auf passende Hardware-Ressourcen als die eigentliche Architekturentscheidung angesehen, in deren Rahmen dem Systemarchitekten Spielraum für die Definition verschiedener Architekturvarianten zusteht. Dabei sind in der Regel eine Vielzahl von Abhängigkeiten und Randbedingungen (Constraints) zu beachten, die den Entwurfsraum frühzeitig einschränken. Trotzdem bleiben für die Realisierung von Systemen eine Vielzahl unterschiedlicher Architekturvarianten möglich, was unter anderem die Grundlage für Produktlinienansätze [CN07] darstellt. Bereits der konzeptionelle Ansatz von AUTOSAR hat zum Ziel, konstruktive Qualitätssicherung als *„Seiteneffekt“* der AUTOSAR Methodik und -infrastruktur zu garantieren:

- Kostenoptimierung durch einfache Wiederverwendbarkeit, Wartbarkeit, Erweiterbarkeit, Portierbarkeit, etc. von Komponenten
- Zuverlässigkeit und Sicherheit durch Wiederverwendung bewährter Komponenten
- Einfache Testbarkeit durch z.B. einheitliche Diagnoseschnittstellen
- Flexible Komponentenverteilung: z.B. zur Reduktion von Buslasten

Im Gegensatz zu den bereits erwähnten konstruktiven Maßnahmen zur *„punktuellen“* Steigerung bestimmter Qualitätskriterien im System sollte auch eine

ganzheitliche, analytische Methodik zur Erfassung und Bewertung von Qualitätskriterien eingesetzt und auf AUTOSAR-Systeme angewendet werden. Das ist insbesondere aus zwei Gründen sinnvoll:

1. Kriterien lassen sich nicht anhand von bestimmten Teilaspekten betrachten, sondern sind nur aussagekräftig, wenn sie auf Grundlage des Gesamtsystems betrachtet werden.

2. Kriterien lassen sich nicht isoliert betrachten und optimieren. Vielmehr können nur im Kontext weiterer Faktoren, insbesondere derjenigen mit gegenläufigen Zielen, geeignete Abwägungen und Kompromisse untersucht werden. Ziel ist demnach, Aussagen über die *„Gesamtqualität“* des Systems zu machen.

Wie in Kapitel 3 beschrieben, kann die Integration einer den Architekturentwurf begleitenden Methodik zur Qualitätssicherung in ein entsprechend erweitertes V-Modell erfolgen.

Der in Abbildung 3.3 dargestellte inkrementelle Prozess sieht eine wiederholte Qualitätsbewertung und anschließende Optimierung des Architekturentwurfs vor, bis ein hinreichende Erfüllung der im Fahrzeugprojekt verfolgten Qualitätsziele für das vollständig implementierte System zu erwarten ist.

Die Modellierung der Systemarchitektur erfolgt zunächst primär in Hinblick auf die umzusetzende Gesamtfunktionalität, wie sie im Rahmen der Analyse der funktionalen Anforderungen ermittelt wurde. Begleitend hierzu können die nichtfunktionalen Anforderungen durch Architektur-Evaluation überprüft werden. Somit wird der Software-Architekt bei den zu treffenden Entwurfs-Entscheidungen unterstützt, indem deren Auswirkungen auf die Qualitätseigenschaften abgeschätzt werden können. Diese Vorgehensweise garantiert zwar keine genauen bzw. absoluten Aussagen über die spätere Qualität, aber sie ermöglicht den Vergleich verschiedener Architekturvarianten sowie das Aufzeigen von Tendenzen und Abhängigkeiten für verschiedener Kriterien, um so geeigneten Tradeoffs für gegenläufige Qualitätsziele ermitteln zu können. Insbesondere für Schlüsselkriterien wie Performanz hat sich gezeigt, dass rein auf Erfahrungswerten basierende Architekturentwürfe häufig nicht zu den am besten geeigneten Varianten führen.

Zur Bewertung einzelner Kriterien auf Grundlage einer Architekturspezifikation müssen im nächsten Schritt Bewertungstechniken bzw. -methoden definiert werden, die eine einheitliche Vorgehensweise zur Abschätzung der zu erwartenden Qualität des späteren Systems definieren. Hier sind vor allem Metriken von Interesse, also Annahmen über Systemeigenschaften, die allein durch *„Messen“* struktureller Eigenschaften auf Grundlage des Architekturmodells getroffen werden. Hierfür können beispielsweise Maße zur Kopplung, Bindung oder Komplexität herangezogen werden, die so verschiedene Bestandteile miteinander in Beziehung setzen. Der Vorteil von Bewertungsfunktionen, die auf Metriken beruhen, besteht

in der vollständig automatisierbaren Auswertbarkeit. Um die Genauigkeit und Aussagekraft von Ergebnissen quantitativer Bewertungstechniken zu erhöhen, ist hingegen zusätzlich die Einbeziehung dynamischer Aspekte notwendig. Hierfür können statistische Verfahren oder Lastprofile herangezogen werden oder auch Simulationsergebnisse, die auf dem Architekturmodell als Prototyp basieren. Derartige Bewertungstechniken sind insbesondere für Kriterien wie Performanz und Verfügbarkeit von Interesse.

Es existieren bereits eine Vielzahl von Techniken zur Bewertung und Analyse spezifischer oder allgemeiner Qualitätskriterien durch Architektur-Evaluation, wie z.B.:

1. Scenario-based Architecture Analysis Method (SAAM)
2. Architecture Trade-off Analysis Method (ATAM)
3. Cost Benefit Analysis Method (CBAM)
4. Architecture-Level Modifiability Analysis (ALMA)
5. Software Performance Engineering (SPE)

Von den vorgestellten Ansätzen [BCK03] erlaubt lediglich ATAM die gleichzeitige Analyse mehrerer Kriterien und deren Abhängigkeiten untereinander, zudem sind in den meisten Verfahren die verwendeten Bewertungstechniken fest vorgegeben.

Für eine umfassende Vorgehensweise zur Qualitätssicherung und -optimierung wollen wir uns deshalb im Folgenden auf den zuvor vorgestellten übergeordneten Ansatz beziehen, der die strukturierte Integration beliebiger Kriterien und Bewertungstechniken zur Evaluation der Gesamtqualität erlaubt [FGB+07]. Dieser Ansatz erlaubt nicht nur den einfachen Vergleich verschiedener Architekturvarianten hinsichtlich ihrer Qualität, sondern auch die Analyse von Abhängigkeiten und Kompromissen verschiedener Qualitätsziele. Insbesondere das Kriterium Kosten kann so gezielt optimiert werden, bei gleichzeitiger Überprüfung und Kontrolle der weiteren Qualitätsziele für das Fahrzeugprojekt.

### Integration von Architektur-Evaluation in die AUTOSAR-Methodik

Soll die zuvor vorgestellte Methodik auf AUTOSAR-konform entwickelte Systeme angewendet werden, ergeben sich folgende Fragestellungen:

1. Welche Rolle spielt der Architekturentwurf in AUTOSAR? In welcher Form ist eine Architektur-Modellierung als formalisierter Entwicklungsschritt innerhalb der AUTOSAR-Methodik integriert? Welche Sichten werden dabei eingenommen? Sind eventuell Anpassungen am zuvor zugrunde gelegten Architekturbegriff notwendig?

2. Kann eine Systemarchitektur in AUTOSAR zur aussagekräftigen Evaluation von Qualitätskriterien verwendet werden? Welchen Einfluss kann der Architektur-Entwurf auf die verschiedenen Qualitätskriterien eines automatisch generierten AUTOSAR-Systems haben? Wie viel Entscheidungsspielraum und Einflussmöglichkeiten hat der Systemarchitekt dabei? Wie viel hängt von den eingesetzten Authoring Tools ab? Wie viel wird durch die Abstraktionsschichten verborgen, insbesondere wenn es um Hardware-nahe Kriterien geht?

3. Können Qualitätskriterien von AUTOSAR-Systemen durch geeignete, aussagekräftige Szenarien charakterisiert werden? Bezüglich welcher Systemartefakte können geeignete Metriken für verschiedene Kriterien definiert werden? Welche Informationen über das System sind im Rahmen des Architekturentwurfs verfügbar?

4. Erlaubt die AUTOSAR Methodik die technische Integration einer automatisierten Architektur-Evaluation in den Entwicklungsprozess? Wie kann eine Anbindung eines Evaluationswerkzeuges an vorhandene Schnittstellen und Werkzeuge erfolgen?

Bevor eine nähere Diskussion dieser Grundfragen erfolgt, kann einleitend bereits festgestellt werden, dass der in AUTOSAR verfolgte Ansatz sehr gut mit der beschriebenen Methodik zusammenpasst, da ein strukturierter Architekturentwurf das Kernstück eines AUTOSAR-Systems darstellt. Nachfolgend wird auf die Fragestellungen näher eingegangen:

**Rolle des Architekturentwurfs in AUTOSAR**

In der AUTOSAR-Methodik und im AUTOSAR-Standard ist ein strikt Komponenten-orientierter Architekturentwurf ein wesentlicher Bestandteil. Dieser bildet innerhalb des Entwicklungsprozesses den Ausgangspunkt für Design und Implementierung des Systems und stellt somit die früheste und grundlegendste Design-Entscheidung mit entsprechend weit reichenden Auswirkungen auf die Eigenschaften des späteren Systems dar. Somit kann der Architekturentwurf als das Kernstück eines zu entwickelnden AUTOSAR-Systems angesehen werden.

Die AUTOSAR-Methodik verfolgt konsequent modellbasierte Methoden zum Entwurf von Systemen. Als Grundlage hierfür definiert der AUTOSAR-Standard ein vollständiges Metamodell zur Spezifikation sämtlicher Systembestandteile, die im Verlauf des Entwurfsprozesses durch den Entwickler festgelegt werden können bzw. durch Authoring Tools automatisch generiert werden. Sämtliche Festlegungen, insbesondere die Architektureigenschaften eines AUTOSAR-Systems, sind somit an fester Stelle in der Metamodell-Ausprägung eines konkreten Projektes in entsprechenden Artefakten abgelegt.

Wie zuvor als Idealvorstellung beschrieben, verfolgt die AUTOSAR-Methodik eine getrennte Betrachtung des Entwurfs von Software- und Hardwarearchitektur, d.h. die Spezifikation der Softwarekomponenten und ihrer Interaktion untereinander im System erfolgt unabhängig von Art und Anzahl der Steuergeräte und der gewählten Netzwerk-Topologie. Erst im letzten Schritt des Architektur-Entwurfs erfolgt die Integration dieser beiden Sichten durch das Mapping von Softwarekomponenten auf Steuergeräte zu deren Ausführung und der daraus resultierenden Abbildung von Datensignalen zwischen Komponenten auf Netzwerksignale.

Der Schwerpunkt von AUTOSAR liegt dabei auf dem Entwurf der Softwarearchitektur, verfolgt also einen flexiblen, funktionsorientierten Ansatz. Zur Unterstützung dieser Vorgehensweise ist das AUTOSAR-Schichtenmodell für Ausführungsplattform darauf ausgerichtet, alle plattform-spezifischen Details über einheitlichen Schnittstellen dem Software-System zu verbergen und so eine flexible Strukturierung sowie eine einfache Wiederverwendbarkeit von Komponenten zu ermöglichen. Weiterhin sieht AUTOSAR eine modellbasierte und durchgängig Werkzeuggestützte Spezifikation der Systemarchitektur vor. Entsprechende Authoring Tools z.B. zur Architekturmodellierung stehen u.a. mit SystemDesk von dSpace, Rational System Developer von IBM oder dem AUTOSAR SystemDesk Pack von IBM zur Verfügung. Diese Ansätze bieten darüber hinaus Möglichkeiten zur frühzeitigen Überprüfung des Systementwurfs bezüglich Fehlerfreiheit, beispielsweise durch modellbasierte Simulation, Algorithmen zur Überprüfung der Vollständigkeit, Konsistenz und Kompatibilität von Modellen (Modellqualität) sowie der automatisierten Code-Generierung.

### Architekturevaluation von AUTOSAR-Systemen

Die Genauigkeit und Aussagekraft der Evaluationsergebnisse hängt wie zuvor schon allgemein beschrieben auch bei AUTOSAR-System grundsätzlich vom auszuwertenden Kriterium und der verwendeten Metriken ab. Bei der Definition der Metriken stellt sich zudem die Frage, welche Informationen der jeweiligen Bewertungstechnik in der Architekturbeschreibung von AUTOSAR zur Verfügung stehen bzw. welche Details durch die Infrastruktur verborgen bleiben. Hierzu können folgende Feststellungen gemacht werden:

- Die Modellierung der Architektur in AUTOSAR erfolgt mit einem hohen Detaillierungsgrad. Weiterhin sind viele Grundkonzepte wie z.B. Kommunikationsprimitive vollständig standardisiert und können bei der Metrikauswertung bereits als *„konstant“* angenommen werden.

- Das AUTOSAR-Metamodell enthält grundsätzlich alle für die Generierung eines Systemrahmens notwendigen Informationen und sollte deshalb auch

hinreichend für die Bewertung der zu erwartenden Qualität des Systems sein.

- Ein Problem besteht hingegen in der strikten Separierung der Applikation (High-Level-SWComponents) von der unterliegenden Infrastruktur (Low-Level-Basic-SWComponents). Hierdurch ergibt sich zwar ein hohes Maß an Flexibilität und Variationsmöglichkeiten beim Entwurf der Systemarchitektur. Allerdings muss für eine hinreichend genaue Evaluation eventuell doch der Ablaufkontext von Komponenten betrachtet werden, also Details zu Runnables und deren Tasks sowie deren Interaktion untereinander und mit der unterliegenden technischen Plattform herangezogen werden. Eine Möglichkeit kann darin bestehen, den aus den Implementierungsmodellen der Komponenten generierten Code in die Architekturmodelle zu Reimportieren und in die Evaluation mit einzubeziehen, wie es beispielsweise SystemDesk ermöglicht.

Die einheitliche Beschreibung von Systementwürfen in AUTOSAR erlaubt den nahtlosen Austausch von Modellfragmenten zwischen Tools der verschiedenen Entwicklungsphasen zur Umsetzung einer lückenlosen, durchgängigen Toolkette. Darüber hinaus kann auf Grundlage des Metamodells und des hierfür definierten UML Profiles eine Integration bestehender UML-Entwicklungsumgebungen in einen solchen Toolverbund erfolgen. Bewährte Techniken aus dem UML-Umfeld, wie beispielsweise umfangreiche Modell-Konsistenzprüfungen (z.B. OCL[1]), Requirements Tracing [Poh08] und Refactoring, können ebenfalls zur konstruktiven Qualitätssicherung des AUTOSAR-Systems beitragen. Genauso kann auf Grundlage dieser Modelle die Definition und Auswertung der für die Architektur-Evaluation benötigten Metriken erfolgen. Das feststehende Metamodell, auf das sich die Methodik stets beziehen kann, erleichtert somit den Vergleich verschiedener Entwurfsvarianten und eine feste Integration eines Werkzeuges zur automatisierten Evaluation in den Entwicklungsprozess. Einmal bewährte Metriken können also stets für beliebige Projekte wiederverwendet werden, solange diese AUTOSAR-konform entwickelt werden. Wie in Punkt 2 und 3 bereits dargelegt, müssen diese Metriken allerdings in Hinblick auf die spezifischen Eigenschaften von AUTOSAR-Systemen abgestimmt und eventuell mit weiteren Erfahrungs- bzw. Expertendaten angereichert werden.

#### Metriken für Qualitätskriterien in AUTOSAR

Grundlage für die Definition von Metriken sind die Artefakte des AUTOSAR-Metamodells, die im Rahmen der Architekturmodellierung durch die hierfür

[1] http://www.omg.org

durch den Standard vorgeschlagenen Sichten in den zugehörigen Diagrammen beschrieben werden:

- SWComponentDiagrams und InternalBehaviorDiagrams für die Software-Architektur
- ECUDiagrams und TopologyDiagrams für die Hardwarearchitektur
- SystemDiagrams für die Systemarchitektur

Auf Grundlage der hierin modellierten Architekturbestandteile können je nach Kriterium entsprechende Metriken definiert werden, z.B.:

- Die Kosten der Steuergeräte können durch Ermittlung der Art (ECUDiagram) und Anzahl (TopologyDiagram) bestimmt werden
- Die Kosten des Leitungsstrangs können durch die Verteilung miteinander interagierender Komponenten im Mapping des SystemDiagrams abgeschätzt werden

Ein Spezialfall sind Kriterien der Modifizierbarkeit, da die Infrastruktur eines AUTOSAR-Systems grundsätzlich beliebig flexibel durch Austausch, Hinzunahme und Entfernen von Komponenten angepasst werden kann. Zudem hängt dieses Kriterium aber auch von der Granularität der Dekomposition der Architektur ab.

Durch das vollständig standardisierte AUTOSAR-Metamodell als invarianter Bezugspunkt für die Metrikdefinition ist zudem gewährleistet, dass bewährte Metriken zur Evaluation beliebiger AUTOSAR-Systeme wieder verwendet werden können. Neben diesen einfachen Beispielen können je nach Bedarf beliebig komplexe Metriken zur Evaluation herangezogen werden und so frühzeitig eine entsprechend genaue Annäherung an die tatsächliche Systemqualität vorgenommen werden.

Im Folgenden sollen einige Beispiele für die Abschätzung der Performanz gegeben werden, verweisen für die Definition von Metriken aber ansonsten auf die entsprechende Fachliteratur.

**Beispiel: Performanz – Prozessorauslastung**

Die spätere Prozessorauslastung ist wesentlich durch das InternalBehavior der Komponenten bestimmt, die auf den jeweiligen Prozessor gemappt werden. Ein Problem bei der Analyse des InternalBehavior besteht dabei in der Entkopplung von Runnables (logische Prozesse) und deren späterer Ausführung durch reale Betriebssystem-Tasks. Die Metriken zur Abschätzung der Prozessorauslastung kann zwar durch das Zeitverhalten der Runnables (RTEEvents) gut angenähert

werden. Bei der späteren Realisierung wird das Laufzeitverhalten allerdings durch die tatsächlichen Tasks sowie das verwendete Schedulingverfahren des Betriebssystems bestimmt, also von Faktoren, die erst in späteren Phasen der Implementierung festgelegt werden. Weitere Elemente aus dem Metamodell, die zur Evaluation herangezogen werden können, sind beispielsweise die Verwendung von ExclusiveAreas, die zu einer Verringerung der Performanz führen. Metriken, die direkt auf Implementierungs-Code ausgewertet werden, könnten ebenfalls hinzugezogen werden, falls dieser z.B. in Form von Legacy-Komponenten zur Verfügung steht.

**Beispiel: Performanz – Buslast**

Eine ähnliche Vorgehensweise kann bei der Ermittlung der Buslast verfolgt werden, die auf Grundlage der Topologie-Beschreibung ermittelt werden kann. Hier ergibt sich das Kommunikationsaufkommen aus der Verteilung von kommunizierenden Komponenten auf verschiedenen Steuergeräten. Auch hier kann die Kommunikationshäufigkeit durch Betrachtung des Zeitverhaltens der Runnables erfolgen.

**Beispiel: Performanz – Speicherauslastung**

Die Speicherausstattung von Steuergeräten ist durch den ECUType bzw. der PerInstance-Memory definiert. Der Speicherbedarf von Komponenten einer ECU ist durch Faktoren bestimmt, wie z.B. die in den Runnables verwendeten Datenelementen oder auch den bei der Kommunikation verwendete Queue-Größen.

Auf Grundlage dieser Überlegungen kann eine Erweiterung der AUTOSAR-Methodik um eine inkrementelle, entwicklungsbegleitende Architekturevaluation zur Qualitätssicherung bei der Definition einer Systemarchitektur zu einem Prozessmodell wie in Abbildung 4.20 dargestellt, führen. Der Schritt der Auswer-

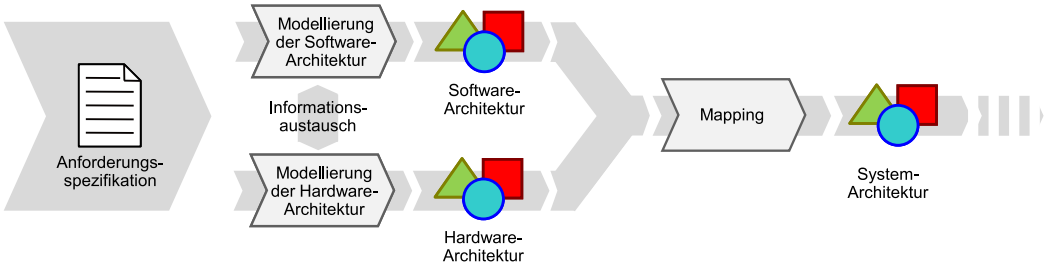


**Abbildung 4.20:** Erweiterte AUTOSAR-Methodik in SPEM-Notation

tung der Einzelkriterien durch geeignete Metriken ist hierbei fest an die Modellierungssichten (AUTOSAR Templates), die darin vorgesehenen Artefakte zur

Architekturbeschreibung und entsprechende Modellschnittstellen gebunden. Diese Systembeschreibungen, die durch entsprechende AUTOSAR Authoring Tools erstellt wurden, sind beispielsweise:

- SoftwareComponentDescription,
- ECUResourceDescription,
- SystemConfigurationDescription,
- CommunicationMatrix,
- etc.

Für eine genauere Evaluation können zudem weitere Artefakte, wie z.B. automatisch erzeugter Code verwendeter Authoring Tools oder auch System Constraints herangezogen werden. Grundsätzlich ist aber zu prüfen, ob die Artefakte Teil des Architekturentwurfes sind bzw. in dieser frühen Phase des Entwurfs schon zur Verfügung stehen.

Auf dieser Grundlage kann schließlich ein iterativer, Qualitätssichernder Architekturentwurf im Rahmen der AUTOSAR-Methodik etabliert und so ein erweiterter Entwicklungsprozess angestrebt werden.

#### Inkrementelle Qualitätsoptimierung

Nachdem eine Evaluation erfolgt ist, kann durch inkrementelle Veränderungen des Architekturmodells versucht werden, die Qualitätsbewertung einzelner Kriterien und unter Umständen des Gesamtentwurfs schrittweise zu verbessern. Je nach eingesetzter Entwicklungsumgebung können entsprechende Assistenzfunktionen eingesetzt werden, um diesen Verfeinerungsprozess zu unterstützten. Dazu zählen zum Beispiel:

- Modell-Refactoring und UML-basierte Techniken des Software-Engineering
- Bewährte Architekturmuster für bestimmte Qualitätsziele
- Versionierung und Verwaltung verschiedener Architekturvarianten
- Requirements-Tracing zur Analyse von Auswirkungen von Änderungen

Darüber hinaus können Analysen oder Heuristiken eingesetzt werden, um gezielt für ein oder mehrere bestimmte Kriterien auf Grundlage der betrachteten Architekturvarianten und deren Evaluationsergebnisse Optimierungspotential aufzuzeigen. Beispielsweise kann durch entsprechende Heuristiken eine ausgewogene Verteilung von Softwarekomponenten auf Steuergeräte ermittelt werden.

Der Erfolg einer Optimierungsmaßnahme lässt sich dann durch erneute Evaluation der neuen Architekturvariante untersuchen. Dabei ist zu beachten, dass Anpassungen am Architekturmodell nicht nur das zu optimierende Kriterium beeinflussen, sondern in der Regel auch Auswirkungen auf die Bewertung anderer Kriterien haben wird und sich somit nicht zwangsläufig eine Verbesserung der Gesamtqualität durch die Optimierungen von Teilkriterien ergeben muss. Beispielsweise kann die Reduktion der Kosten durch Entfernen von Steuergeräten gleichzeitig zur Verschlechterung der Performanz führen, da sich nun die Software-Funktionen auf weniger Rechenressourcen verteilen.

Somit stellt die Verbesserung der Gesamtqualität ein multikriterielles Optimierungsproblem mit inhärent gegenläufigen Zielen und einer Vielzahl von Randbedingungen dar. Eine Automatisierung dieses Vorgangs ist somit kaum praktikabel, vielmehr kommt Erfahrungswerten auch zukünftig eine große Bedeutung zu. Speziell für den hier betrachteten Evaluationsansatz können zudem durch Sensitivitätsanalyse innerhalb der Graphenstruktur Optimierungspotentiale offen gelegt werden [FGB+07].

Gerade im Rahmen des AUTOSAR-Ansatzes, der prinzipiell eine beliebig flexiblen Verteilung von Softwarekomponenten auf Steuergeräte zulässt, besteht großer Spielraum, um verschiedenste Architekturvarianten zur Optimierung der Qualität zu erproben. Insbesondere die Art, Anzahl, Vernetzung und Komponentenzuordung der Steuergeräte im System haben zukünftig einen wesentlichen Einfluss auf Schlüsselkriterien wie Kosten, Prozessorlast, Buslast, Speicherbedarf, Modifizierbarkeit, etc.

### Werkzeugunterstütung

Um den zuvor beschriebenen iterativen Prozess zur Qualitätsoptimierung von AUTOSAR-System im Rahmen des Architekturentwurfes zu realisieren, ist eine Werkzeugunterstützung zur Automatisierung bzw. Unterstützung der Evaluation und allen dazugehörigen Schritten als integraler Bestandteil der bestehenden Werkzeugverbünde notwendig. Im Folgenden sollen mögliche Ansätze hierfür diskutiert und anschließend eine prototypische Umsetzung auf Basis der Fallstudie vorgestellt werden.

Ein Werkzeug zur Integration der Qualitätssicherungsmaßnahmen basierend auf Architektur-Evaluation wie zuvor beschrieben, soll die Ausführung folgender Aktionen zur Definition von Szenarien für Qualitätskriterien unterstützen:

1. **Definition von Metriken:** Spezifikation einer Metrik-Sprache, mit der Modell-Artefakte adressiert und korreliert werden können. Zusätzlich sollen qualitative Bewertungstechniken unterstützt werden, die Eingaben von statistischen Werten, Simulationsergebnissen, allgemeine Experteneinschätzungen etc. ermöglichen. Diese Informationen könnten entweder am Modell

selbst annotiert werden oder auch über ein eigenes Beschreibungsformat zum Zeitpunkt der Evaluation eingebracht werden.

2. **Definition von Qualitätsraten:** Diese können wie zuvor beschrieben beliebige Verläufe annehmen und könnten in Form von einfachen Wertetabellen bis hin zu komplexen Intervall-funktionen beschrieben werden.

3. **Einlesen einer Systemarchitektur:** Hierfür müssen geeigneten Schnittstellen (z.B. COM oder XML) sowie eine zum AUTOSAR-Metamodell passende Datenstruktur vorhanden sein.

4. **Automatisierte Auswertung der Evaluationsstruktur:** Implementierung eines Algorithmus zum Einlesen und Auswerten komplexer Bewertungsmetriken, Normalisierung durch die Qualitätsrate sowie zur Bottom-Up-Integration der Teilergebnisse zu einer Gesamtqualität. Schließlich müssen die Ergebnisse geeignet auf einer grapfischen Oberfläche aufbereitet werden.

Abbildung 4.21 zeigt die Architektur eines prototypischen Werkzeuges zur Umsetzung der beschriebenen Schritte. Neben diesen Grundfunktionen können weitere

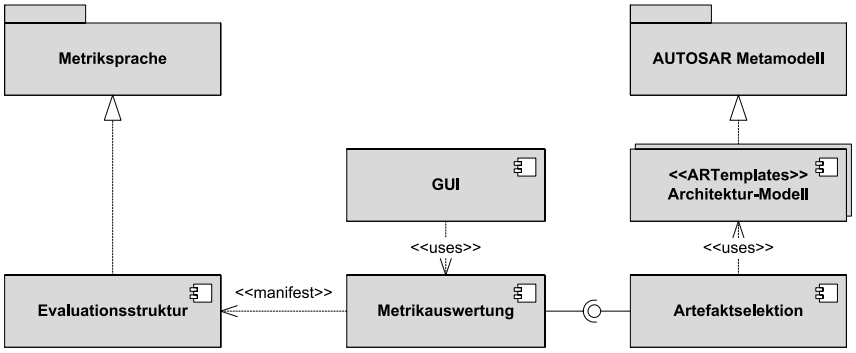


**Abbildung 4.21:** Bestandteile des Evaluationswerkzeuges

Techniken zum Einsatz kommen, um eine schrittweise, iterative Optimierung der Systemarchitektur zu erzielen, z.B.:

- Requirements Tracing zu Modellartefakten und zugehörigen Requirements, die für die Evaluationsergebnisse verantwortlich sind und zur Optimierung des Ergebnisses beitragen können,

- Unterstützung von Modell-Refactoring, um anhand der Evaluationsergebnisse durch Umstrukturierungen die Qualität des Architekturmodells zu optimieren,

- Darstellung der Unterschiede verschiedener Varianten sowie Vorschläge zu Optimierungspotentialen
- etc.

### Fallstudie zur Architektur-Evaluation

Das GUI des realisierten Werkzeugs, das die zuvor beschriebenen Evaluationsschritte umsetzt, ist in Abbildung 4.22 zu sehen. Die Evaluationsstruktur für

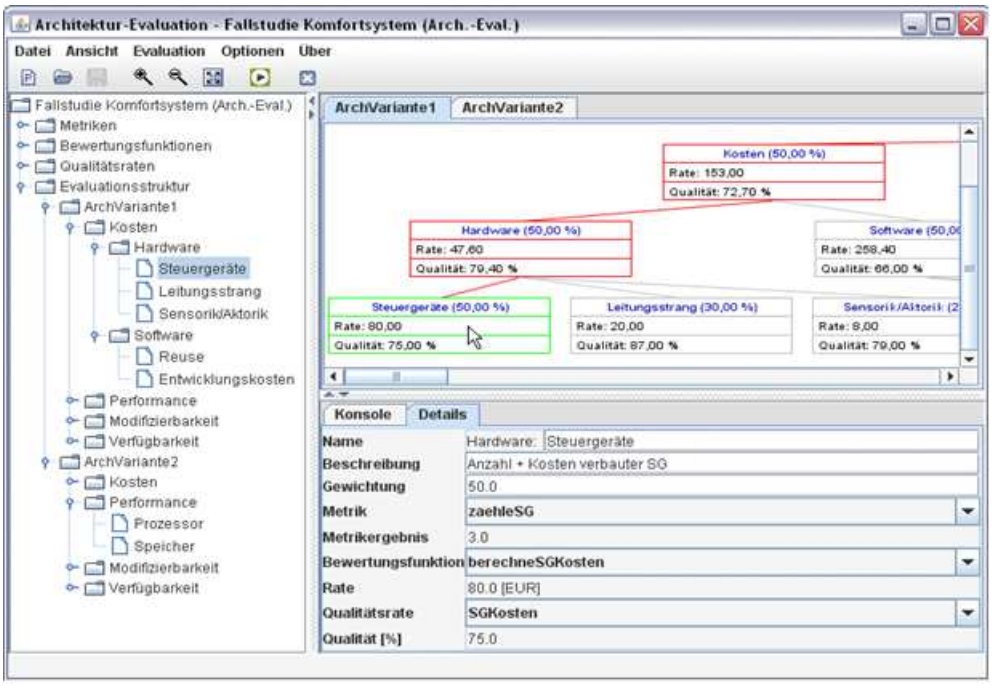


**Abbildung 4.22:** GUI des Evaluationswerkzeuges

die Qualitätsanforderungen der Fallstudie ist im zentralen Bereich des GUIs zu sehen. Eine genauere Beschreibung erfolgt später.

Das Einlesen der zu evaluierenden AUTOSAR-Architektur kann auf verschiedene Arten erfolgen:

- Einbettung in ein UML-Werkzeug wie z.B. das Rhapsody AUTOSAR Pack, dass das AUTOSAR UML Profile zur Modellierung von AUTOSAR-Systemen verwendet. Zur Selektion relevanter Modell-Artefakte kann dann beispielsweise OCL verwendet werden.
- Einlesen aus AUTOSAR XML-Dateien, deren festes Schema ebenfalls Teil des AUTOSAR-Standards ist und auch durch Modellierungswerkzeugen

wie z.B. PREEvision von Aquintos[2] exportiert werden können, die keine reinen AUTOSAR Auhtoring Tools sind. Die Selektion relevanter Modell-Artefakte kann zum Beispiel über XQuery[3] erfolgen.

- Verwendung der COM-Schnittstelle eines Authoring-Tools.

Die vorgestellte prototypische Implementierung verfolgt letztgenannten Ansatz, in dem die Modellelemente über die Python-Schnittstelle des Architekturmodellierungswerkzeugs SystemDesk selektiert werden. Es hat sich gezeigt, dass diese Variante der Anbindung zweckmäßig ist. Die Definition und Auswertung von Metriken kann entweder direkt implementiert (hard coded) werden, oder es kommt eine Metrik-Sprache mit entsprechenden Korrelationsoperatoren für verschiedene Modellelemente zum Einsatz. Bei der Festlegung der Syntax und Auswertungssemantik dieser Sprache muss beachtet werden, dass die Ausdrucksstärke (Mächtigkeit bzw. Expressiveness) hoch genug für den angestrebten Genauigkeitsgrad der Evaluation ist. In dem entwickelten Evaluationswerkzeug wurden die Metriken zunächst direkt implementiert, eine Erweiterung um spezifische Metrik-Sprachen ist für zukünftige Anwendung möglich.

In der vorgestellten Fallstudie, ein PKW Komfortsystem, wurde das Evaluationswerkzeug in den Toolverbund integriert. Es wurden dabei verschiedene Architekturvarianten für die Fallstudie anhand von exemplarischen Qualitätskriterien evaluiert. Die Evaluationsstruktur für die in der Fallstudie exemplarisch definierten Qualitätsziele ist in Abbildung 4.23 zu sehen und wurde entsprechend im Werkzeug umgesetzt und ausgewertet. Als Bewertungsfunktionen kamen größteneils Metriken zum Einsatz, die im Werkzeug direkt in Java-Code auf Grundlage von Modellabfragen realisiert wurden. Beispielsweise erfolgt die Bewertung der Steuergeräte-Kosten durch Selektion der in Hardware-Topologie verbauten Steuergeräte und anschließendem Aufsummieren der durch sie verursachten Kosten. Für die Kriterien zur Modifizierbarkeit wurden eine Reihe von Szenarien untersucht und der resultierende Anpassungsaufwand für die Architektur qualitativ abgeschätzt.

Je nach Variante ergaben sich unterschiedliche Bewertungsergebnisse im Bereich zwischen 40% und 80%, wobei vor allem der Faktor Kosten aufgrund der hohen Gewichtung entscheidend für das Evaluationsergebnis war. Der exemplarische Verlauf der Qualitätsrate für das Kriterien Kosten ist in Abbildung 4.24 dargestellt. Im Werkzeug können Qualitätsraten durch entsprechende Intervallfunktionen spezifiziert werden.

Die zuvor vorgestellte Architekturvariante unter Verwendung von vier Steuergeräten stellte sich als ein gut geeigneter Tradeoff zwischen den verfolgten

[2] http://www.aquintos.com/
[3] http://www.w3.org/XML

Qualitätskriterien dar und wird nachfolgend als Grundlage für das Design und die Implementierung des Komfortsystems verwendet.

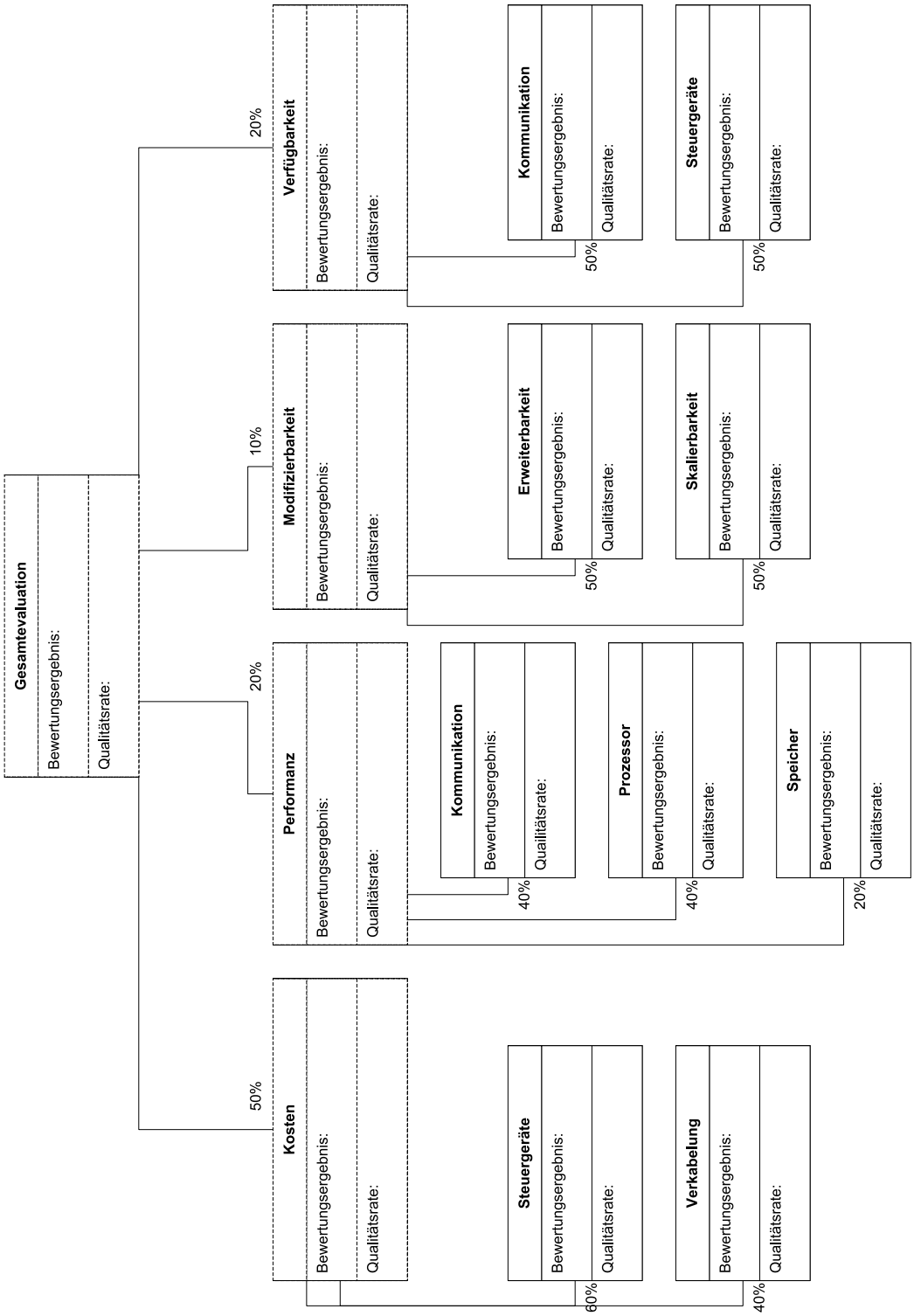


**Abbildung 4.23:** Evaluationsstruktur der Fallstudie

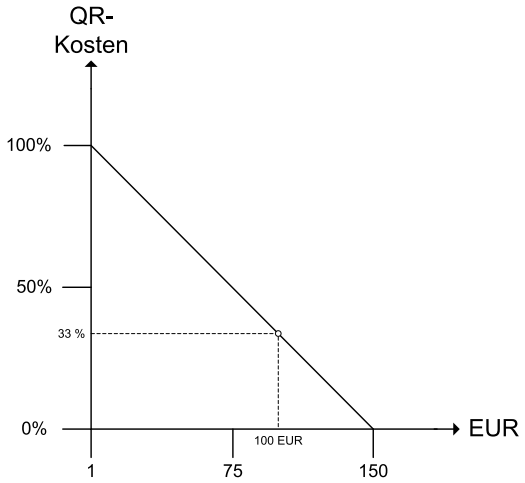


**Abbildung 4.24:** Qualitätsrate des Kriteriums Kosten

## 4.4 Design und Implementierung

Ausgehend vom vorhergend angestellten Architekturentwurf wurden in dieser Phase des V-Modells die Funktionen für die vier Steuergeräte in Programmcode umgesetzt. Die Funktionen für die drei Steuergeräte Türsteuergerät, Zentralverriegelungssteuergerät und Alarmanlagensteuergerät wurden mit den Softwarepaket MATLAB/Simulink/Stateflow modelliert und mit automatischer Codegenerierung in C-Code überführt. Die Funktionen für das HMI-Steuergerät wurden direkt in C umgesetzt. Die Details hierzu sowie die Konfiguration und die Verwaltung von AUTOSAR-Projekten werden in den folgenden Abschnitten ausgeführt.

### 4.4.1 Projektstruktur und -verwaltung

Nach Abschluss des Systementwurfs, und damit dem Mapping von Softwarekomponenten auf Steuergeräte, ist die Implementierung eines einzelnen Steuergerätes im Prinzip in zwei Bereiche aufgeteilt. Die Entwicklung der Softwarekomponenten auf der Anwendungsebene stellt ein Bereich dar. Der zweite ist die Konfiguration der Basis-Software mit automatisierter Code-Generierung. Das Gesamtsystem besteht letztlich aus dem Quellcode beider Ebenen. Zur Erzeugung der Binärdateien für den ROM-Bereich der Steuergeräte muss der gesamte Quellcode compiliert und gelinkt werden. Abbildung 4.25 zeigt die prinzipielle Aufteilung der Projektebenen für ein Steuergerät auf Basis von AUTOSAR. Die AUTOSAR-Softwarekomponenten bilden die Applikationsebene und die RTE, die Basis- und MCAL-Module bilden die Basisebene.

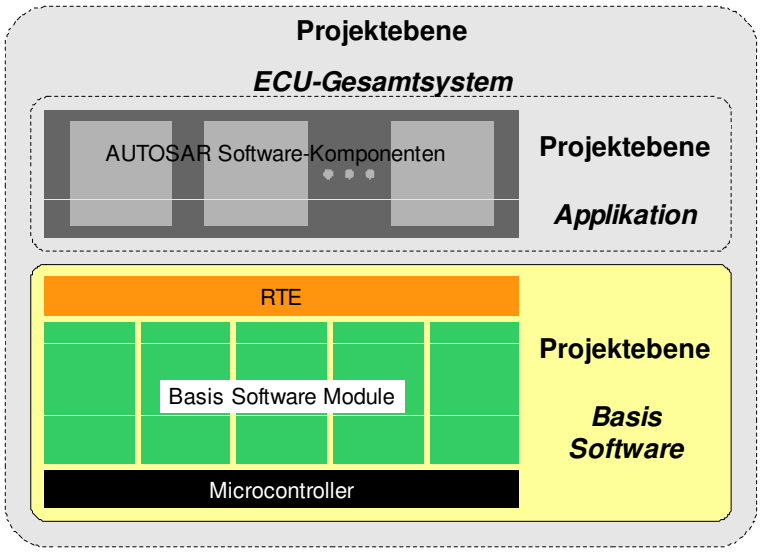


**Abbildung 4.25:** Projektebenen eines ECU-Gesamtsystems

Die Projektverwaltung für die Ebene der Basis-Software ist in tresosECU enthalten. Über die Bedienoberfläche können Projekte für verschiedene Steuergeräte angelegt werden. Diese Projekte enthalten eine Auswahl von Basismodulen, die für den Betrieb des Steuergerätes vorgesehen sind. Auf einen einzigen Befehl hin kann tresosECU die Konfiguration aller Basismodule eines Projektes prüfen oder alle erforderlichen Quelldateien der Basis-Software des Steuergerätes generieren (siehe Abschnitt 4.5).

Für die Applikationsebene oder für das ECU-Gesamtsystem war eine komfortable Projektverwaltung bei der Durchführung dieser Fallstudie nicht vorhanden. Als Hilfsmittel zur Compilierung der Quelldateien und zum Verlinken diente das klassische *make*. Für die Basismodule erzeugt tresosECU die entsprechenden Makefiles, während diese für das Gesamtprojekt einschließlich der Applikationsebene manuell bearbeitet werden mussten. Als Basis für ein Projekt konnten Make-Dateien von Beispielprojekten, die sich im Lieferumfang von tresosECU befinden, herangezogen und modifiziert werden.

### 4.4.2 Implementierung mit C

AUTOSAR-Softwarekomponenten können grundsätzlich in der Programmiersprache C erstellt werden. Für viele Bereiche bei der Entwicklung von Steuergeräte-Software kann die klassische Programmierung in C gegenüber der modellbasierten Entwicklung Vorteile bieten oder unumgänglich sein. Bei der Adaption von bereits entwickelten Lösungen in C auf AUTOSAR kann das Beibehalten der Programmiersprache das Auftreten von Fehlern vermindern und die Anpassungszeit minimieren. Aus MATLAB/Simulink/Stateflow-Modellen erzeugt TargetLink Quellcode in C ebenso wie tresosECU C-Code für die konfigurierten Basis-Module generiert. Im Folgenden wird beschrieben, welche prinzipiellen Regeln für die Erstellung von AUTOSAR-Softwarekomponenten in C gelten. Eine umfassende Dokumentation ist unter [AUT07o] zu finden.

Die Methodologie von AUTOSAR sieht die Dekomposition in atomare Softwarekomponenten vor, die auch bei der Entwicklung in C gilt. Die Softwarekomponenten stellen meist komplexe Einheiten dar, die aus Funktionseinheiten bzw. Prozessen zusammengesetzt sind, den sogenannten Runnables. Ein Runnable kann durch singuläre Ereignisse, wie z.B. das Eintreffen einer Nachricht, oder durch periodische Ereignisse, wie Timing Events, gestartet werden. Es ist möglich, auch verschiedene Events vorzusehen. Die Konfiguration der Events erfolgt nicht im C-Code, sondern in der XML-Beschreibung der Softwarekomponenten (siehe Abschnitt 4.5. Im Folgenden wird auf ausgewählte Implementierungsspezifischen Details näher eingegangen.

### Runnables

Das Auslösen und die Kommunikation der Runnables wird durch die RTE vollzogen. Es gibt sie in den Kategorien 1a, 1b und 2. Runnables der Kategorie 1a und 1b müssen in endlicher Zeit beendet werden und dürfen keine Wartepunkte besitzen. Eine Runnable der Kategorie 1a kann selbst keine Kommunikationsaufrufe tätigen, d.h. die RTE übernimmt den Datenaustausch vor und nach Ausführung der Runnable. In der Kategorie 1b ist der Empfang und das Senden während der Ausführung der Runnable möglich. Die Kategorie 2 erlaubt die Nutzung von Wartepunkten.

In C wird eine Runnable als eine Funktion ohne Rückgabewert und Argumente definiert, die einen beliebigen Namen haben darf, z.B. void ExampleRunnable(){...} . Der Name wird in die XML-Beschreibung eingetragen und bei der RTE-Konfiguration einem Task zugewiesen.

### Kommunikation

Der Austausch von Nachrichten mit anderen Softwarekomponenten wird über die RTE realisiert, auch wenn sich die Komponenten auf dem gleichen Steuergerät befinden. Das Senden von Nachrichten bei einem Sender-Receiver Interface erfolgt mit dem Aufruf

```
status = Rte_Send_<Portname>_<Datenelement> ( data );
```

oder

```
status = Rte_Write_<Portname>_<Datenelement> ( data );
```

Rte_Send wird benutzt, wenn für die Kommunikation eine gepufferte Übertragung vereinbart wurde (Flag isQueued = true), Rte_Write gilt für die ungepufferte Übertragung. Im Namen des Funktionsaufrufes ist sowohl der Name des Ports als auch der Name des Datenelements enthalten. Diese beiden Angaben müssen mit den Definitionen in der XML-Beschreibung übereinstimmen. Viele RTE-Aufrufe liefern als Rückgabewert einen Statuscode, der angibt, ob und gegebenenfalls welche Fehler bei der Ausführung aufgetreten sind.

Für den Empfang von Nachrichten bei einem Sender-Receiver Interface ist der Funktionsaufruf als

```
status = Rte_Receive_<Portname>_<Datenelement> ( &data );
```

oder

```
status = Rte_Read_<Portname>_<Datenelement> ( &data );
```

definiert. Auch hier gilt die Unterscheidung zwischen gepufferter und ungepufferter Übertragung. Rte_Receive gilt für den ersten Fall (Flag isQueued = true) und Rte_Read für den zweiten. Auch diese Funktionen liefern Statuscodes als Rückgabewert. Als Argument wird ein Zeiger auf das zu empfangene Datenelement übergeben.

Empfängt eine Runnable Daten aus verschiedenen Quellen, spielt die Auswertung der Rückgabewerte eine wichtige Rolle. Wird die Runnable in Folge eines Datenempfangs- oder Timing-Events aufgerufen, sollte für jedes empfangbare Datenelement der entsprechende RTE-Aufruf erfolgen. Anhand des Rückgabewertes kann festgestellt werden, ob das Datenelement empfangen wurde oder nicht. Folgendes Beispiel zeigt dieses Prinzip:

```
void ExampleRunnable()
{
 DataType1 data1 = 0;
 DataType2 data2 = 0;
 /* Abholen von Daten von R_FirstPort/DataValue1 erfolgreich? */
 if (Rte_Receive_R_FirstPort_DataValue1( &data1 ) == RTE_E_OK)
 { /* Ja */ ...
 }
 /* Abholen von Daten von R_SecondPort/DataValue2 erfolgreich? */
 if (Rte_Receive_R_SecondPort_DataValue2( &data2 ) == RTE_E_OK)
 { /* Ja */ ...
 }
}
```

Das Symbol RTE_E_OK hat die Bedeutung „Kein Fehler aufgetreten“.

### I/O-Zugriffe

Lesen oder Schreiben von Hardware-Ports erfolgen über ein Client-Server Interface. Der RTE-Aufruf für das Einlesen eines I/O-Ports lautet

```
status = Rte_Call_<Portname>_OP_GET ( &data );
```

Das Schreiben eines I/O-Ports erfolgt mit

```
status = Rte_Call_<Portname>_OP_SET ( data );
```

Auch hier geben die zurückgelieferten Statuswerte Aufschluss über erfolgreiche oder fehlgeschlagene Ausführung.

Zur vollständigen Implementierung von I/O-Zugriffen gehören Definitionen in XML-Beschreibungen und das Erstellen einer spezifischen Softwarekomponente, welche die I/O Hardware Abstraction beinhaltet. Diese Komponente besteht aus C-Code und einer XML-Definition. Jeder genutzte Port muss in der I/O Hardware Abstraction angelegt werden.

### Verwendung der Programmiersprache C++

Nach der AUTOSAR-Spezifikation ist es prinzipiell zulässig, Software-Komponenten in der Programmiersprache C++ zu erstellen. Beim Einsatz von C++ auf Steuergeräten sind einige grundsätzliche Aspekte zu beachten. Oftmals besit-

zen Steuergeräte eine vergleichsweise geringe Speicherkapazität und andere eingeschränkte Ressourcen. Daher muss bei der Auswahl der Programmiersprache beispielsweise deren Speicherverwaltung besonders berücksichtigt werden. Die Sprache C++ begünstigt eine dynamische Speicherverwaltung, die auf Steuergeräten zu Problemen führen kann. „In der Embedded-Programmierung wird dynamische Speicherverwaltung eher zurückhaltend verwendet. Das hat hauptsächlich damit zu tun, dass aufgrund der Fragmentierung des Speichers, die mit der Zeit auftritt, die Operationen nicht deterministisch ablaufen. Die Implementierungen von *new* und *delete* können zwar, je nach Compilerhersteller, besser sein als ihre C-Gegenüber *malloc* und *free*, aber auch hier wird der Speicher im Laufe der Zeit fragmentiert.“ [KMS08]. Daher sollte bei der Programmierung von Steuergeräten der Speicher statisch organisiert werden. Um dies gewährleisten zu können, ist ein fundiertes Wissen über die Mechanismen der Speicherverwaltung von C++-Code zur Laufzeit erforderlich. Bestimmte C++-Konstrukte sollten nicht verwendet werden.

Ein weiterer Aspekt ist die Eignung der aus dem C++-Code erzeugten Binärdateien für den nicht-flüchtigen Speicher (ROM) des Steuergerätes. Einige Compiler und Tools unterstützen die automatische Umsetzung. Ohne diese Hilfe müssen im Code Datentypen und Konstrukte benutzt werden, die der jeweiligen Speicherstruktur des Steuergerätes angepasst sind.

Im C++-Code sollte auf das Exception handling (Ausnahmenbehandlung) verzichtet werden, da dies oft mit einem hohen Ressourcenverbrauch verbunden ist und häufig eine dynamische Speicherverwaltung verwendet wird.

Der Einsatz der Sprache C++ zur Programmierung von Steuergeräten ist mit einigen Einschränkungen möglich. Vorausgesetzt werden in jedem Fall grundlegende Kenntnisse darüber, wie der Compiler C++-Konstrukte umsetzt und wie kritisch sich diese zur Laufzeit im Steuergerät verhalten können. Im Einzelfall bleibt abzuwägen, ob die Einschränkungen in Kauf genommen werden können und der Einsatz von C++ gerechtfertigt ist.

Gleiche Rahmenbedingungen gelten prinzipiell auch für den Einsatz von C++ zur Erstellung von AUTOSAR-Softwarekomponenten. Nach Angaben des Herstellers Elektrobit ist es möglich, C++-Code in tresosECU-Projekten zu verwenden. Inwieweit dies mit den Tools in der Praxis umgesetzt werden kann, wurde im Rahmen dieser Fallstudie nicht untersucht.

### Umsetzung des HMI-Steuergeräts

Die Funktionen des HMI Steuergeräts wurden in der Programmiersprache C umgesetzt. Der Software-Aufbau des HMI basiert auf vier Funktionseinheiten. Eine Einheit dient zur Auswertung der Drehknopf- und der Tastersignale. Die Ausgangssignale des Drehknopfes werden abgetastet und mittels einer Auswerte-

logik in Events für die interne Weiterverarbeitung umgewandelt. Ein Event steht für einen Schritt im Uhrzeigersinn, der andere Event für die Gegenrichtung. Zwei weitere Events werden bei Druck auf die Tasten ESC und ENTER erzeugt.

Ein weiterer Funktionsblock dient dem Nachrichtenempfang. Darin werden eingehenden Daten für beispielsweise Außentemperatur, Fahrzeuggeschwindigkeit und Alarmauslösung entgegengenommen und der zentralen Verarbeitungseinheit zugeführt.

Die zentrale Funktionseinheit reagiert auf die eingehenden Events und Daten. Die Drehknopfbewegungen und Tastendrücke werden in Navigationsschritten durch die Einstellungsmenüs und Statusanzeigen umgewandelt. Die Menüstruktur wird in Abbildung 4.26 schematisch dargestellt.

Die vierte Funktionseinheit dient zum Versenden von Nachrichten, wie z.B. den vom Nutzer veränderten Parametern für die anderen Steuergeräte.

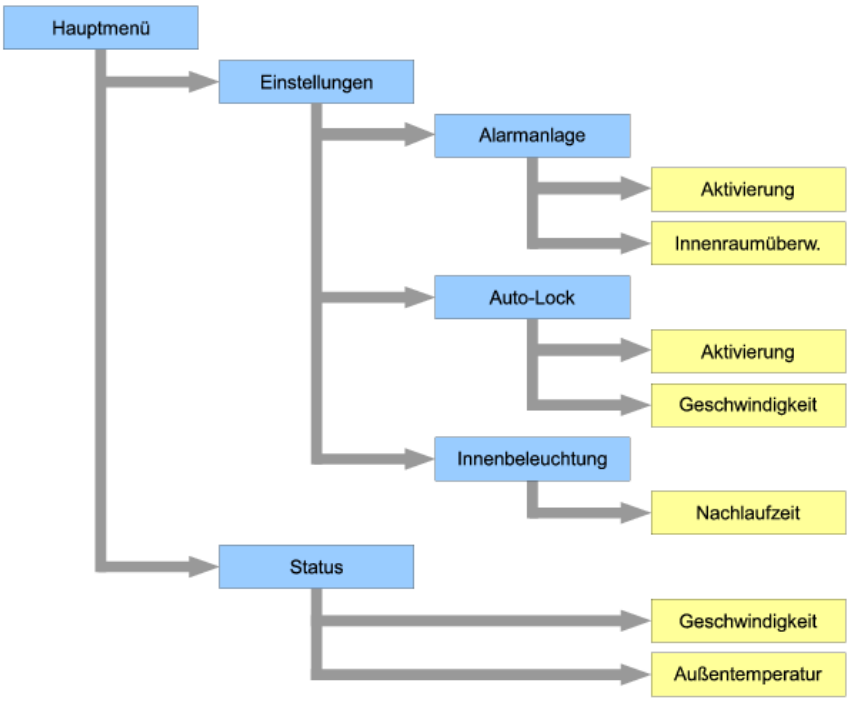


**Abbildung 4.26:** HMI Menüstruktur

### 4.4.3 Debugging von Steuergeräte-Software

Zum Debuggen von Steuergeräten sieht der AUTOSAR-Standard verschiedene Möglichkeiten vor. Die Basis-Software von AUTOSAR stellt das Modul Development Error Tracer (Det) zur Verfügung, das zur Behandlung von Fehleraus-

gaben dient. Das Modul selbst kann keine Fehler erkennen, sondern nur die vom Entwickler eingesetzten Funktionsaufrufe Det_ReportError() weiterverarbeiten. Es ist für die Entwicklungsphase der Komponenten der Basis-Software konzipiert und lässt sich für verschiedene Arbeitsweisen konfigurieren. Die Fehlerbehandlung kann durch Schreiben der Fehlermeldung in einen Ringpuffer im RAM, durch Stoppen des Programms (Breakpoint) oder durch den Aufruf einer Callback-Funktion erfolgen.

Sofern das zur Entwicklung eingesetzte Steuergerät ein On-Chip Debug System (OCDS) aufweist, besteht die Möglichkeit, dies mit einem komfortablen Hardware-Debugger zu nutzen. Für den TriCore TC1766 Mikrocontroller bietet beispielsweise die Firma Lauterbach GmbH das Debug-System TRACE32 an, welches an einen Pfostenverbinder des Entwicklungsboards angeschlossen wird. Mit TRACE32 können unter anderem Programmablauf und Prozessorregister vollständig überwacht, Haltepunkte gesetzt und Prozessorbefehle schrittweise abgearbeitet werden. Das Debuggen ist sowohl im Assembler-Code als auch im C-Quelltext möglich und das Flash-ROM des Steuergerätes kann mit diesem System programmiert werden.

Einen ähnlichen Funktionsumfang bieten Debug-Umgebungen verschiedener Hersteller für das sogenannte „Wiggler Interface“ des TriCore 1766 Entwicklungs-Boards. Diese Debug-Schnittstelle wird über ein einfaches Kabel mit dem Parallelport eines Personal Computers verbunden und ist im Lieferumfang des Entwicklungs-Boards enthalten.

Im Rahmen dieser Fallstudie wurden drei Debugger erprobt. Der CrossView Pro Debugger vom Hersteller Tasking/Altium ist in der Entwicklungs-Software für den TriCore 1766 Mikrocontroller enthalten und auch für die „Wiggler“-Schnittstelle geeignet. Während sich Beispiel-Code aus dem Programmpaket von Tasking/Altium damit debuggen ließ, akzeptierte das Tool die ELF-Binärdateien mit AUTOSAR-konformem Code nicht. Auch der HiTOP Debugger der Firma Hitex GmbH unterstützt das „Wiggler Interface“. Aufgrund von wiederholten Übertragungsfehlern bei der Nutzung der Schnittstelle erwies sich auch dieses Tool als ungeeignet. Die Untersuchung des TRACE32-Debuggers verlief durchweg positiv. Alle oben genannten Funktionen konnten fehlerfrei genutzt werden.

### Debug-Ausgaben über die serielle Schnittstelle

Eine vergleichsweise einfache Methode zur Überwachung von Programmabläufen ist die Anzeige von Warnungen, Fehlermeldungen oder Zuständen. Für diese Ausgaben bietet sich beim TriCore-Entwicklungs-Board die serielle Schnittstelle ASC0 an. Über diese Schnittstelle wird das Flash-ROM des Steuergerätes mit der Software MemTool von Infineon programmiert. Die Kabelverbindung mit dem Entwicklungs-PC steht nach dem Programmiervorgang für das Debuggen

zur Verfügung, für das lediglich ein Terminal-Programm mit Unterstützung des COM-Ports auf dem PC benötigt wird.

Zur Ausgabe der Debug-Meldungen über die serielle Schnittstelle wurden Routinen aus Programmierbeispielen von Infineon angepasst und eingebunden.

Der Aufruf von dprintf() kann in den C-Code von Softwarekomponenten oder Basismodulen geschrieben werden. Dies gilt auch für den automatisch generierten C-Code aus MATLAB/Simulink-Modellen oder für den von tresosECU generierten Quellcode der Basis-Software. Bedingt durch die langsame Übertragungsrate der seriellen Schnittstelle sollten häufige Debug-Ausgaben vermieden werden, da sie zu starken Verzögerungen im Programmablauf führen können.

### 4.4.4 Implementierung mit Matlab/Simulink/Stateflow

Das Software-Paket mit den Komponenten MATLAB/Simulink/Stateflow und der Erweiterung TargetLink zur Codegenerierung für AUTOSAR dient zur modellbasierten Entwicklung von Steuergeräte-Software. Die Software MATLAB wird von der Firma The MathWorks entwickelt und vertrieben. Der Softwarename steht für MATrix LABoratory. Die Stärken von Matlab liegt in der numerischen Berechnung von Matrixen, sowie der Datenanalyse und Visualisierung. MATLAB dient als Basiskomponente für das Zusatzpaket Simulink, das die Modellierung und Simulation von zeitkontinuierlichen und zeitdiskreten Systemen erlaubt. Simulink bietet eine grafische Bedienungsoberfläche und stellt für verschiedene Anwendungsgebiete Bibliotheken bereit, die sich individuell erweitern lassen. Stateflow ist eine Erweiterung von Simulink zur Erstellung und Einbindung von Zustandsautomaten und Ablaufdiagrammen. Simulink und Stateflow stammen ebenfalls wie die Basissoftware MATLAB von The MathWorks.

Die modellbasierte Software-Entwicklung, wie sie z.B. mit Simulink durchgeführt werden kann, bietet Vorzüge für eine Vielzahl von Anwendungsgebieten. Neben der Funktionsmodellierung kann auch das mechanische System modelliert werden. Durch die sich daraus ergebenden Simulationsmöglichkeiten können die zu entwickelnden Modelle bereits vor der Verfügbarkeit der Hardware in der Simulation erprobt und angepasst werden. Die modellbasierte Entwicklung von Steuergerätfunktionalitäten findet bereits seit einigen Jahren Einzug bei den Automobilherstellern [Hor05, MHH+03, CFGK05]. Simulink unterstützt von vorherein die hierarchische Strukturierung des gesamten Systems und vereinfacht damit die Beherrschung komplexer Projekte. Die Lesbarkeit und Verständlichkeit von Projekten wird durch diese hierarchisch geordneten Modelle gefördert.

### Umsetzung des Alarmanlagensteuergerät

Auf dem Alarmanlagensteuergerät werden mehrere Funktionen realisiert. Dies sind zum einen die Alarmanlage selbst, zum anderen aber auch die Innenraumbeleuchtung, die Steuerung der Blinker vorne sowie die Tasten für die Zentralverriegelung. Die oberste Modellebene der hierarchischen Implementierung des Alarmanlagensteuergeräts in Simulink, in der die Einzelfunktionen zu erkennen sind, ist in Abbildung 4.27 dargestellt.

Im Folgenden soll als Beispiel die detaillierte Implementierung der Alarmanlagenfunktion in Stateflow beschrieben werden (Abbildung 4.28). Insgesamt besitzt die Alarmanlage zehn Zustände, die sich weiter strukturieren lassen. Es wird zum einen grundsätzlich unterschieden, ob die Innenraumüberwachung aktiviert oder deaktiviert ist. Zusätzlich zu dieser Unterscheidung wird zwischen den Zuständen „Alarmanlage aus“, „Alarmanlage an“ und „Alarm aktiv“ unterschieden. Wurde der Alarm ausgelöst und nicht bereits während der 20 Sekündigen Alarmphase quittiert, wartet die Alarmanlage in „Alarm, Quittierung ausstehend“ auf die Quittierung. Hierbei entfällt die zuvor genannte grundsätzliche Unterscheidung zwischen Innenraumüberwachung aktiviert oder deaktiviert. Die hier vorliegende Musterlösung lässt sich noch vereinfachen, da zur Demonstration weitere hilfreiche Funktionen implementiert wurden, die in den Anforderungen aber nicht gefordert waren. Hierzu gehört z.B. die Bewegungserkennung von Objekten im Fahrzeug trotz deaktivierter Innenraumüberwachung, die in der vorliegenden Implementierung wie gefordert keinen Alarm auslöst, aber zusätzlich mit Hilfe einer LED signalisiert wird.

### Umsetzung des Türsteuergerät

Auf dem Türsteuergerät werden die Funktionen für die Ansteuerung des Fensterhebers als auch die Spiegeleinstellung realisiert.

Der Fensterheber hat in der vorliegenden Fallstudie die Besonderheit, dass ergänzend zum mechanischen Fenster mehrere neben dem Fenster angebrachte LEDs die aktuelle Fensterhöhe anzeigen. Die Realisierung der Fensterheberfunktionalität zeigt Abbildung 4.29. Insgesamt wurde die Funktionalität mit fünf Stateflow Charts realisiert. Diese lassen sich grob in „Fensterinitialisierung, Fensterbewegung und Einklemmschutz“ sowie „Positionsbestimmung und LED Anzeige“ einteilen. Die Notwendigkeit der Fensterinitialisierung ergibt sich dadurch, dass sowohl für den Einklemmschutz als auch für die LED Anzeige die maximal obere und minimal untere Position des Fensters bekannt sein muss. Als weiteres Beispiel ist in Abbildung 4.30 die Implementierung der Anforderung das Fenster herauf- oder herunterzufahren in Stateflow dargestellt. In einem nachfolgenden Statechart werden die Ausgangsgrößen dieses Charts noch priorisiert z.B. unter Berücksichtigung des Einklemmschutzes.

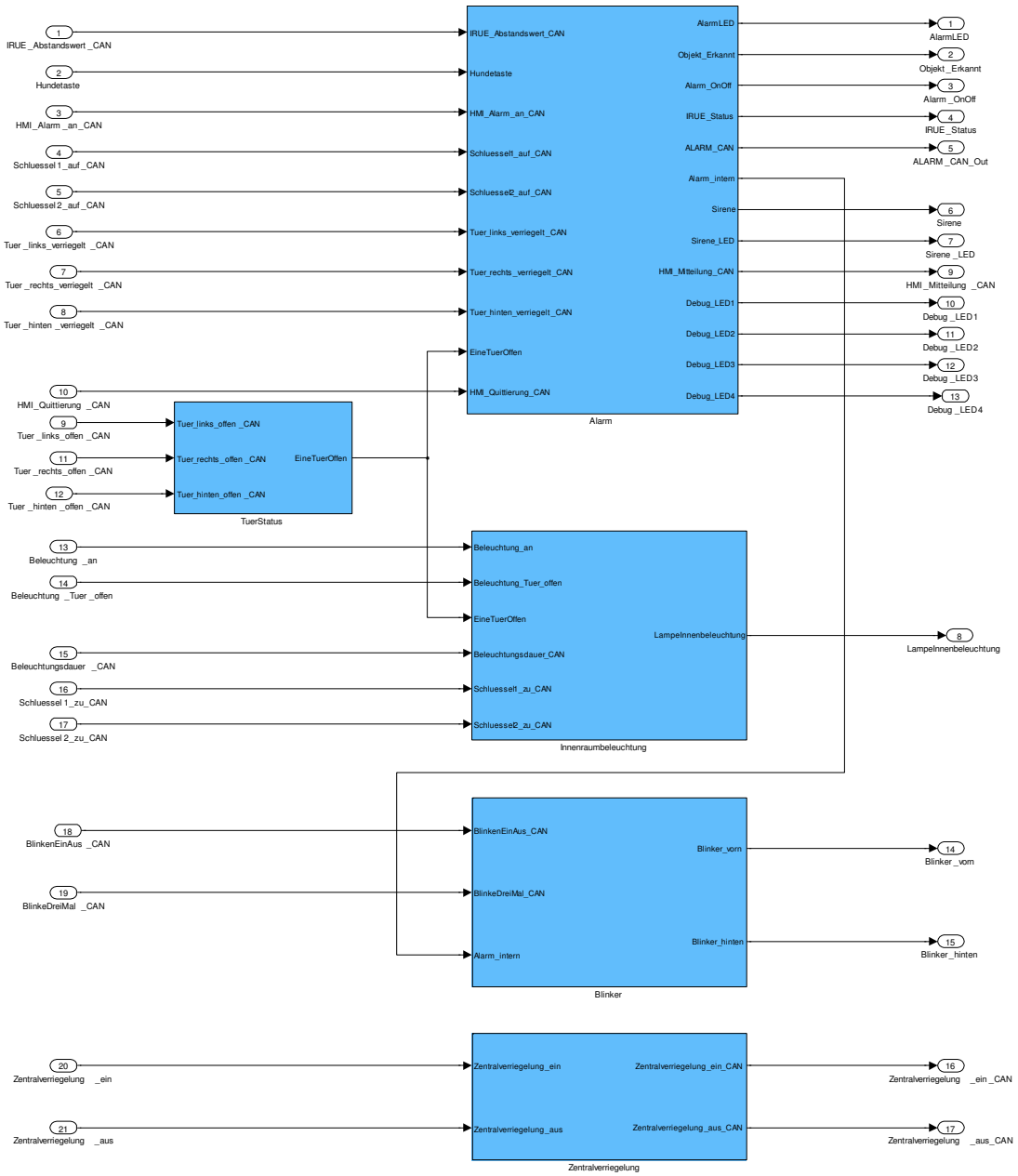


**Abbildung 4.27:** Oberste Modellebene des Alarmanlagensteuergeräts

Die Spiegeleinstellung kann über zwei Informationen gesteuert werden. Zum einen kann die Spiegelposition über Taster am Steuergerät eingestellt werden. Zum anderen wird die Spiegelposition in Abhängigkeit des Fahrzeugschlüssels beim Verschließen des Fahrzeuges gespeichert und beim Aufschließen des Fahr-

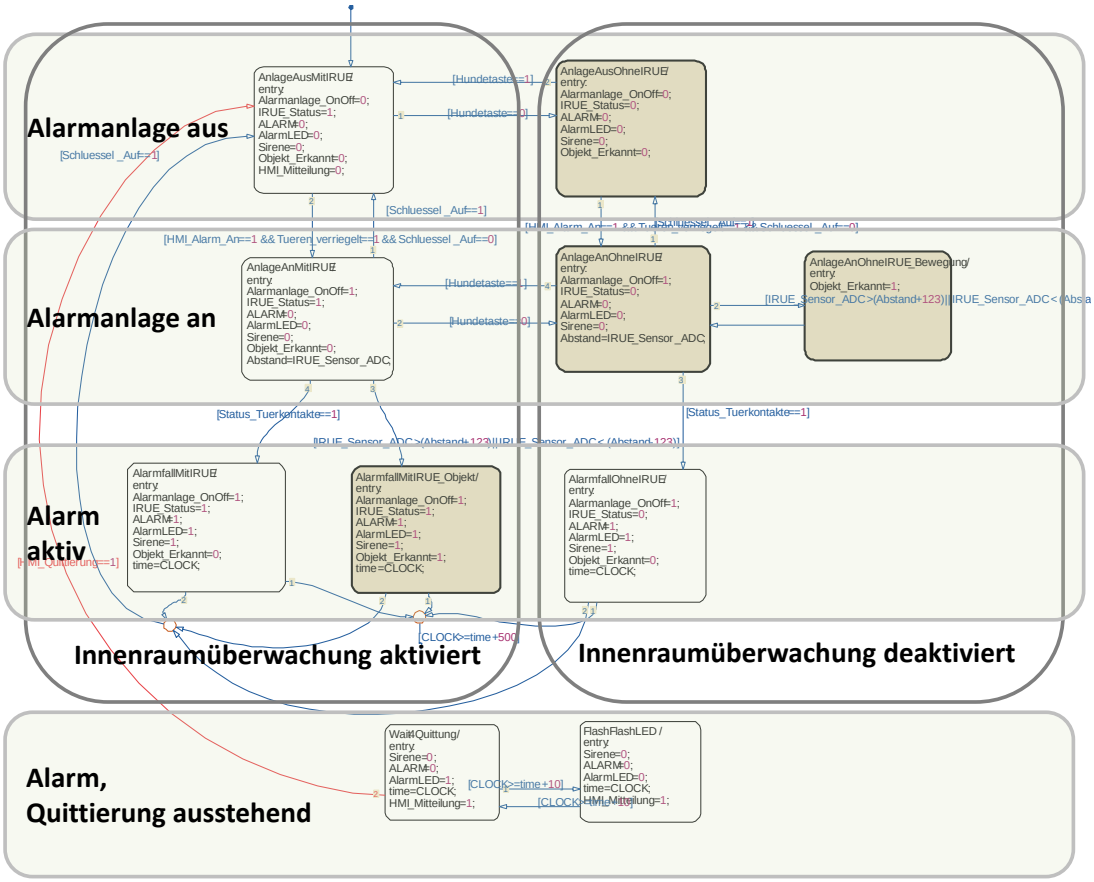


**Abbildung 4.28:** Umsetzung der Alarmanlagenfunktionalität in Stateflow

zeuges werden die Spiegel wieder in die gespeicherte Position verfahren. Ergänzend zur Spiegel- und Fenstersteuerung werden auf dem Steuergerät auch die Blinker in den Außenspiegeln angesteuert.

### Umsetzung des Zentralverriegelungssteuergerät

Auf dem Zentralverriegelungssteuergerät ist das Funkmodul der Türschlüssel, die Schlossaktorik als auch die mechanischen Türkontakte nachgebildet. Die Implementierung in Stateflow in Abbildung 4.31 zeigt die Komplexität die in den Statecharts abgebildet werden kann. Die dargestellte Stateflow-Implementierung realisiert die Verriegelung und Entriegelung der Türen, wobei beim Schließvorgang der Status der Türen (d.h. mechanisch geschlossen oder offen) berücksichtigt werden muss. Aufgrund der vielen unterschiedlichen Zustände, die das Fahrzeug

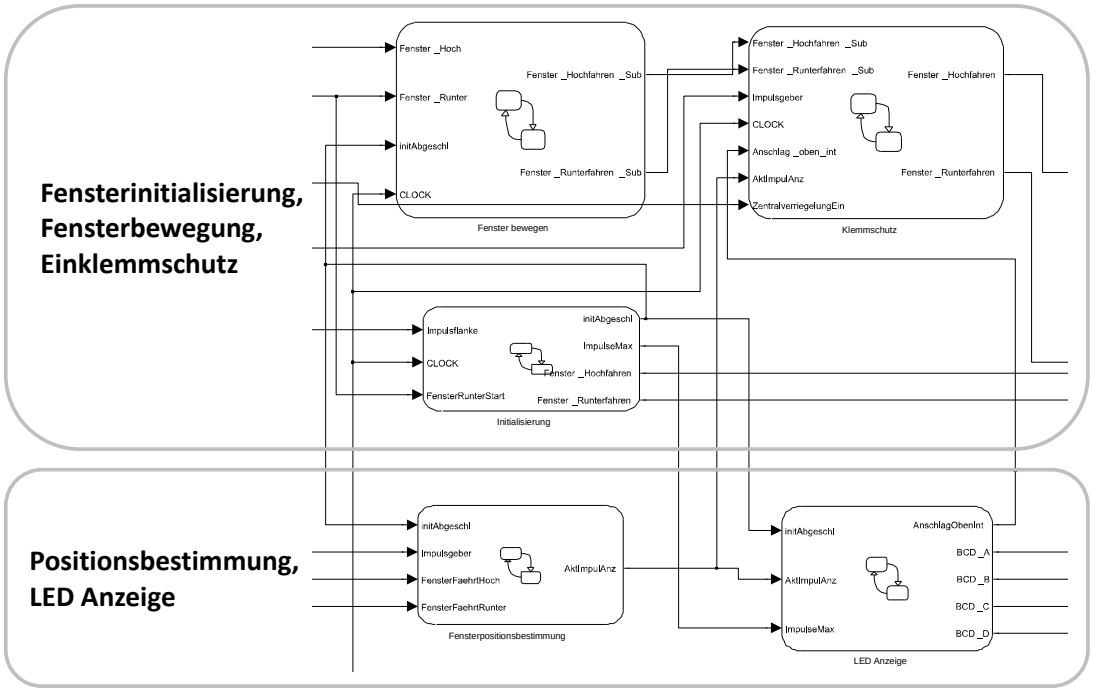


**Abbildung 4.29:** Umsetzung der Fensterhebersteuerung in Simulink

bei der Anforderung das Fahrzeug zu verschließen haben kann, ergibt sich die umfangreiche dargestellte Implementierung.

### 4.4.5 Codegenerierung und TargetLink

Aus den mit Simulink erstellten Modellen kann zusätzliche Software Programmcode erzeugen, um sie z.B. auf Mikrocontrollern nutzen zu können. Im Rahmen der Fallstudie wurde das Zusatzpaket TargetLink des Herstellers dSPACE eingesetzt. TargetLink erzeugt aus Simulinkmodellen automatisch Programmcode für eine Vielzahl von Mikrocontrollern und kann darüber hinaus C-Code für AUTOSAR-Systeme generieren. Hierzu integriert sich TargetLink in die Entwicklungsumgebung von MATLAB/Simulink und stellt eine spezielle Bibliothek bereit. Ein großer Vorteil der modellbasierten Entwicklung ist die Hardware-Unabhängigkeit und die daraus resultierende Wiederverwendbarkeit der Modelle.

Zur Verwendung von TargetLink ist die vorherige Konfiguration des Modells im so genannten Data Dictionary notwendig. Beim Data Dictionary handelt es sich um einen zentralen Datencontainer, in dem z.B. Parameter- und Variablendefinitionen für das Modell und die Code-Generierung abgelegt werden. In diesem Data Dictionary wurden zum einen die verwendeten Softwarekomponenten, zum

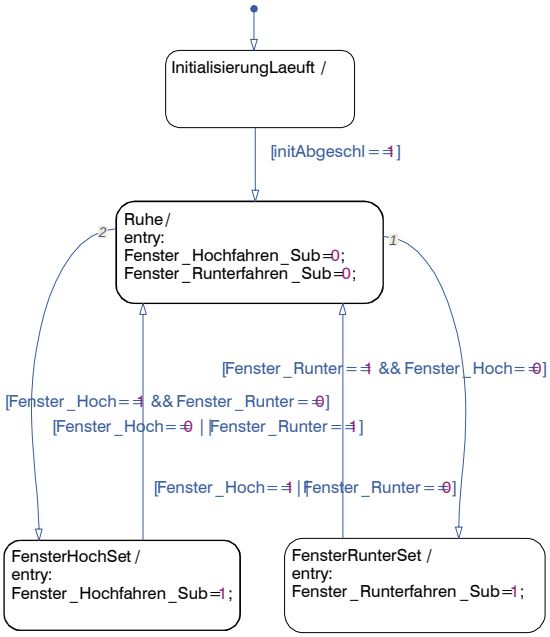


**Abbildung 4.30:** Ansteuerung Hoch- und Runterfahren Fensterheber in Stateflow

anderen die Interfaces, die für den Austausch von Daten zwischen Softwarekomponenten und der Außenwelt notwendig sind, sowie die Runnables spezifiziert. Eine weiterführende Beschreibung folgt in Abschnitt 4.4.6.

Für eine AUTOSAR-konforme Modellimplementierung in Matlab/Simulink sind auf oberster Ebene die zwei Blöcke TargetLink Main Dialog und das Subsystem TargetLink Model-in-the-loop mode aus der TargetLink-Bibliothek notwendig (Abbildung 4.32). Über den TargetLink Main Dialog wird nach vollständiger Spezifikation im Data Dictionary und Erstellung aller Funktionsmodelle das Erzeugen der XML- und C-Dateien gestartet. Dabei beschreiben die C-Dateien den funktionalen Aufbau, und die XML Dateien den strukturellen Aufbau mit den Schnittstellen nach außen. Im Subsystem-Block befinden sich die eigentlichen Softwarekomponenten. Die Definition des Gesamtsystems, das mehrere Softwarekomponenten enthalten kann, wird ebenfalls in den XML-Dateien festgehalten (Abschnitt 4.4.7).

### 4.4.6 Das Data Dictionary

Eine besondere Bedeutung kommt der Konfiguration des Modells im Data Dictionary zu, einem zentralen Datencontainer. Vor der Platzierung von AUTOSAR-spezifischen Elementen im Modell müssen vorausgehend die notwendigen Defini-

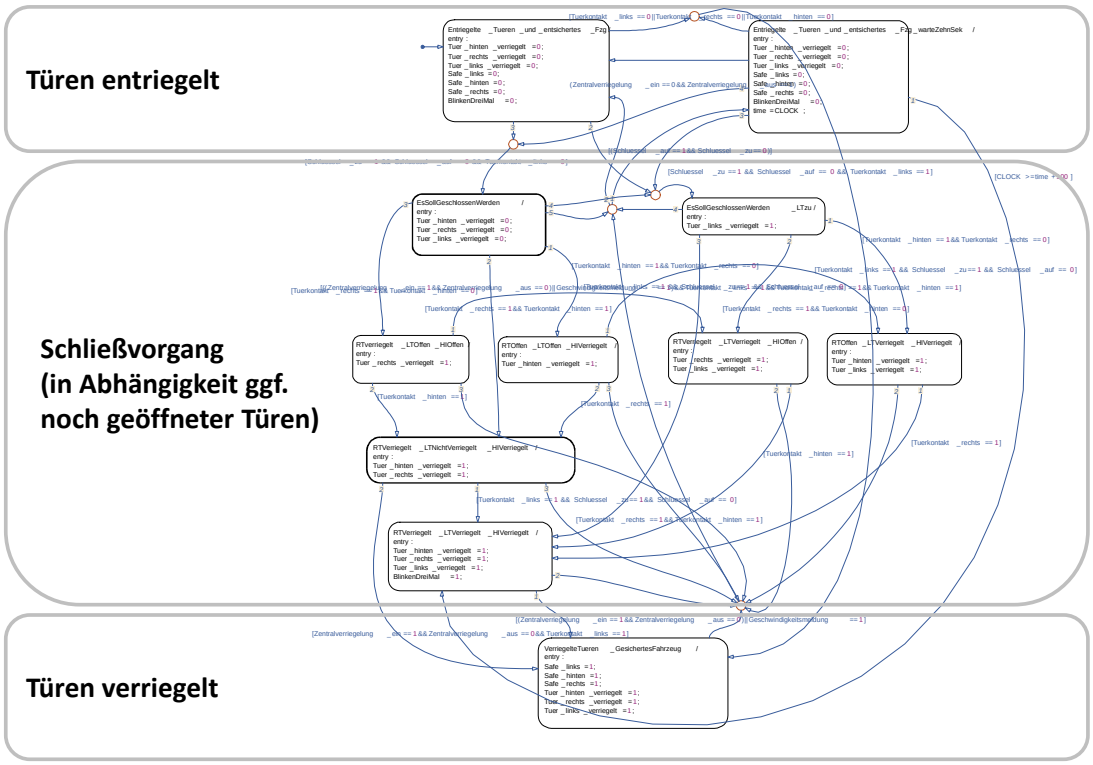


**Abbildung 4.31:** Umsetzung der Schlossaktorik in Stateflow

tionen im Data Dictionary eingetragen werden. Als Ausgangsbasis für ein Modell diente die vordefinierte Vorlage

%DSPACE_ROOT%\Dsdd\Config\dsdd_master_autosar.dd

Darin sind z.B. Variablen, Typendefinitionen, Klassen und Strukturen für AUTOSAR enthalten [dSP06]. Eine neue Softwarekomponente wird im Data Dictionary unter

/Pool/AUTOSAR/SoftwareComponents/<Name der Softwarekomponente>

angelegt. Die Eigenschaft *SupportsMultipleInstantiation* sollte auf „off“ gesetzt werden. Innerhalb der neuen Softwarekomponente werden dann die Ports, Runnables und RteEvents definiert. Es ist sinnvoll, zuvor die Interfaces zu erstellen [dSP06], da in den Untergruppen der Softwarekomponenten darauf Bezug genommen wird.

### Anlegen der Interfaces

Interfaces liegen im Data Dictionary unter *DataDictionary/Pool/Interfaces*.

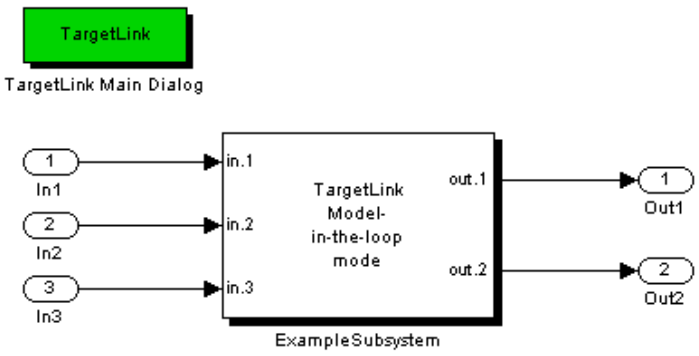


**Abbildung 4.32:** TargetLink Basiskomponenten

Es wird grundsätzlich zwischen Nachrichten- und Port-I/O-Interfaces unterschieden. Die Definition der Interfaces für Nachrichten kann sich an im Modell enthaltenen Softwarekomponenten orientieren, d.h. jede Softwarekomponente erhält ein Interface. Eine Besonderheit stellt die Kommunikation zwischen einzelnen Softwarekomponenten dar.

### Interfaces für Nachrichten

Zunächst wird der Fall betrachtet, dass die Softwarekomponenten mit der Außenwelt, z.B. anderen ECU, Nachrichten austauschen sollen. Im Data Dictionary wird unter */Pool/Interfaces* ein neues SenderReceiverInterface angelegt. In den Eigenschaften (Property Value List) sollte das Feld *IsService* auf „off“ eingestellt werden. Dann wird für jede Nachricht ein neues DataElement eingefügt.

Im Feld *Description* sollte eine aussagekräftige Beschreibung des Nachrichteninhalts stehen, das Feld *IsQueued* sollte auf „on“ gesetzt werden und in Type sollte ein RTE-kompatibler Typ ausgewählt werden. Für jede Nachricht, die die Softwarekomponente senden oder empfangen soll, wird ein eigenes DataElement angelegt. Die Namen der Elemente dürfen eine Länge von 32 Zeichen nicht überschreiten, da ansonsten Fehler im späteren Verlauf der Tool-Kette entstehen. Die Zeichenlänge im Feld Description unterliegt keiner solchen Beschränkung.

### Interfaces für I/O-Ports

Bei Zugriffen auf I/O-Ports waren zum Zeitpunkt der Erstellung einige Besonderheiten zu berücksichtigen. Die verwendete Vorgehensweise hat sich zum Zeitpunkt der Erstellung der Modelle als praktikabel erwiesen, sie muss aber als ein Workaround angesehen werden. Im Data Dictionary wird zuerst für je-

de Datenrichtung ein ClientServerInterface eingerichtet. Im von TargetLink und tresosECU generierten C-Code lauten die Funktionsaufrufe dann

Rte_Call_<Name des Input Ports>_OP_GET()

bzw.

Rte_Call_<Name des Output Ports>_OP_SET()

Auf der Ebene der generierten XML-Dateien war eine Verbindung der I/O-Ports der Modelle mit denen der I/O-Hardware-Abstraction-Komponente in der Compositions-Datei nicht möglich. Trotz identischer Definition meldete tresosECU den Fehler, die Interfaces seien nicht kompatibel. Ein Ausweg war letztlich der Workaround, die Namen der I/O-Port-Interfaces in der von TargetLink erzeugten Softwarekomponentenbeschreibung mit Hilfe eines Texteditors zu ersetzen.

### Interfaces für I/O-Port-Busse

Die beiden Interfaces für den Zugriff auf I/O-Ports dienen zum Lesen und Schreiben jeweils eines einzelnen Port-Bits. Sollen Zugriffe auf Port-Busse stattfinden, müssen neue Interfaces in der entsprechenden Breite definiert werden. Unter der Eigenschaft Type wird statt Boolean die gewünschte Größe, z.B. UInt32 eingestellt. Sind Zugriffe auf mehrere Busse verschiedener Breite gefordert, muss entweder für jede Breite (8, 16, 32 Bit) ein eigenes Interface angelegt oder das Interface mit der größten Busbreite benutzt werden. Die entsprechenden Anpassungen müssen sowohl im XML- als auch im C-Code der I/O-Hardware-Abstraction-Komponente vorgenommen werden.

### Anlegen der Ports für eine Softwarekomponente

Nach Erstellen aller erforderlichen Interfaces folgt die Definition der Ports unter

/Pool/Autosar/SoftwareComponents/<Name der Softwarekomponente>/Ports/

Ein Port für eingehende Nachrichten wird als Receiver Port und ein Port für ausgehende Nachrichten wird als Sender Port angelegt. Ports für I/O-Zugriffe werden unabhängig von der Datenrichtung immer als Client Ports definiert. In den Eigenschaften (Property Value List) wird die SenderReceiverInterfaceRef bzw. die ClientServerRef dann auf das dazugehörige Interface gesetzt. Die Auswahl des DataElements erfolgt an anderer Stelle.

### Anlegen der Runnables und RTE Events

Es ist empfehlenswert, die Runnables einer Softwarekomponente im Data Dictionary einzutragen, bevor sie im Modell platziert werden. Zu den Runnables

gehören RTE Events [dSP06]. Sie legen fest, wann eine Runnable ausgeführt wird. Im vorliegenden Fall wurden nur zwei Arten verwendet, das Timing Event und das Data Received Event. Die RTE Events werden als eigene Gruppe wie Ports und Runnables unter einer Softwarekomponente angelegt.

### Aufbau eines Modells mit zwei Softwarekomponenten

Am Beispiel eines Modells mit zwei Softwarekomponenten sollen elementare Prinzipien aufgezeigt werden. Das Modell beinhaltet den Empfang und das Senden von Nachrichten sowie das Lesen und Schreiben von Ports. Im Beispiel kommunizieren die beiden Softwarekomponenten miteinander, d.h. Komponente 1 sendet eine Nachricht an Komponente 2. Abbildung 4.33 zeigt die zweite Ebene des Modells. Die erste Ebene entspricht der Abbildung 4.32. Die beiden Komponenten empfangen jeweils eine Nachricht und lesen jeweils einen Port. Komponente 1 schreibt einen Port und sendet Daten an Komponente 2. Komponente 2 sendet Daten nach außen.

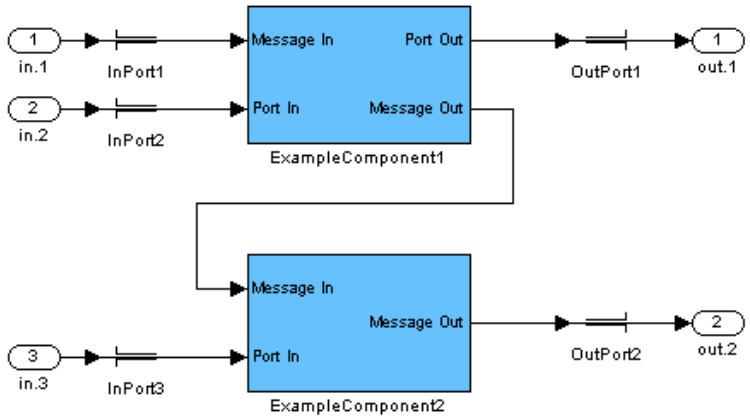


**Abbildung 4.33:** Modell mit zwei Softwarekomponenten

### Beispiel für eine Softwarekomponente

Der Empfang von Nachrichten erforderte zum Zeitpunkt der Erstellung besondere Vorkehrungen, um eine fehlerfreie Verarbeitung zu gewährleisten. Für jeden Eingangs-Port von Nachrichten muss eine eigene Runnable vorgesehen werden. Diese Runnable wird ausschließlich durch den Data Received Event des zugehörigen DataElements ausgelöst. Sie führt keine Verarbeitungsschritte aus, sondern stellt die empfangenen Daten der nachfolgenden Runnable zur Verfügung.

Abbildung 4.34 zeigt den Aufbau einer Softwarekomponente mit zwei Blöcken, die jeweils eine Runnable beinhalten. Der Block Data Received Runnable dient wie beschrieben zum sicheren Empfang von Daten. Im Block Timing Event Runnable läuft der eigentliche Verarbeitungsprozess in einem konstanten Intervall ab.

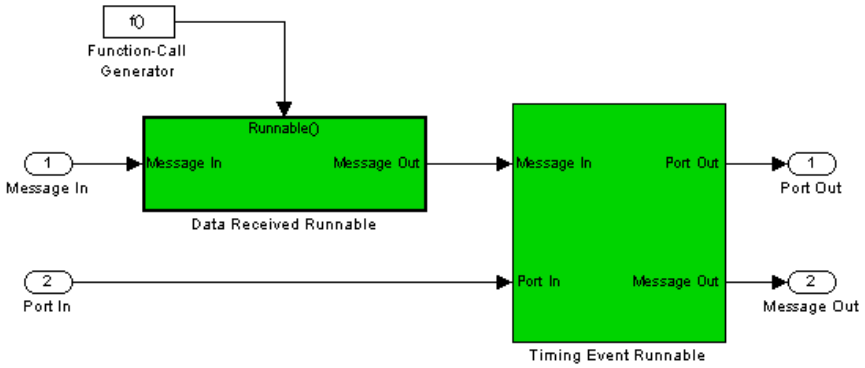


**Abbildung 4.34:** Aufbau einer Softwarekomponente mit Nachrichtenempfang

Abbildung 4.35 zeigt die Anordnung der Ein- und Ausgangs-Ports der zeitlich gesteuerten Runnable. Zum Lesen eines Input Ports wird der Block Runnable Inport aus der TargetLink/AUTOSAR-Bibliothek platziert und auf den zuvor im Data Dictionary spezifizierten Port konfiguriert. Analog wird zum Schreiben eines Output-Ports der Block Runnable Outport gesetzt und konfiguriert. Das gleiche Element wird auch für Ports zum Versenden von Nachrichten verwendet. Über die Konfiguration erfolgt die Zuordnung zum Sender-Port inklusive Interface und DataElement. Auf die gleiche Weise kann in einer zeitlich gesteuerten Runnable ein Block zum Empfang von Nachrichten eingesetzt werden.

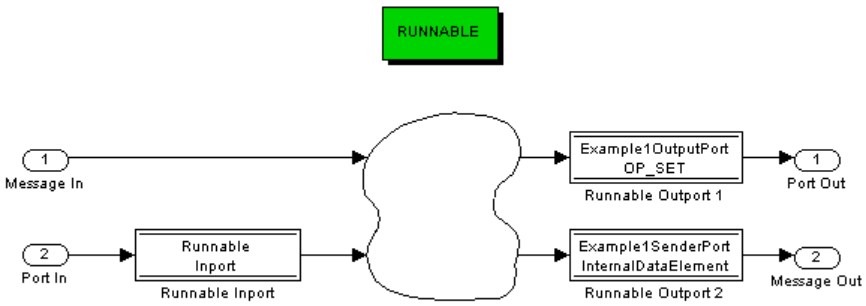


**Abbildung 4.35:** Aufbau der „Timing Event Runnable“

### 4.4.7 Systembeschreibung durch XML-Dateien

Verschiedene Bestandteile eines AUTOSAR-Systems, wie z.B. Software-Komponenten, werden in Dateien im XML-Format [Dau03] beschrieben. Die Dateien können maschinell erstellt und gelesen werden. TargetLink erzeugt neben C-Code auch eine Software Component Description als XML-Datei, welche die Softwarekomponenten nach einem AUTOSAR-konformen Schema beschreiben. Aus diesem Schema müssen Definitionselemente zusammen mit Beschreibungen von System-Komponenten und Verbindungen zwischen Ports in einer sogenannten Top Level Composition [AUT07c] im XML-Format eingetragen werden. Die Top Level Composition wird von tresosECU als Systembeschreibung des zu konfigurierenden Steuergerätes ausgewertet. Eine Composition ist letztlich auch als Komponente aufzufassen und kann selbst Bestandteil einer anderen Composition sein.

Abbildung 4.36 zeigt als Beispiel einen Auszug aus einer Top Level Composition. In der Darstellungsform Grid wird die hierarchische XML-Struktur mit Hilfe von verschachtelten Datenbehältern veranschaulicht. Zur besseren Übersicht sind einige der Datenbehälter geschlossen. In der oberen Hälfte unter Components/Component-Prototype sind alle Komponenten mit der Angabe aufgelistet, welche Anschlüsse sie bereitstellen oder erfordern. Die Softwarekomponente IoHwAbsPrototype stellt Ports bereit (Provided-Com-Specs). Die Komponenten von Nummer fünf bis neun besitzen Ports, die verbunden werden müssen (Required-Com-Specs). Unter Connectors/Assembly-Connector-Prototype in der unteren Hälfte sind alle Verbindungen zwischen Ports auf der obersten Ebene (Top Level) aufgelistet. In diesem Beispiel sind zehn Verbindungen enthalten. Die Anschlusspunkte jeder Verbindung sind in den zugehörigen Datenbehältern Provider-Iref und Requester-Iref definiert.

Abbildung 4.37 zeigt beispielhaft die grafische Darstellung der Top Level Composition eines Steuergerätes mit den entsprechenden Anschlusssymbolen, wie sie in AUTOSAR definiert sind [AUT07c]. Einige der erforderlichen Anschlüsse (required Ports) an der Unterseite der Software-Komponenten sind in der Darstellung nicht verbunden. Diese Ports werden am Virtual Functional Bus angeschlossen, dessen Aufgabe die RTE im Steuergerät übernimmt. Die direkten Verbindungen (Assembly Connectors) zwischen einzelnen Softwarekomponenten eines Steuergerätes sind zwar zulässig, stehen aber prinzipiell dem Grundansatz von AUTOSAR entgegen, die Komponenten nur über den Virtual Functional Bus zu verbinden, um z.B. ein hohes Maß an Flexibilität bei der Verteilung der Softwarekomponenten auf mehrere Steuergeräte gewährleisten zu können. Im Rahmen dieser Fallstudie war es allerdings notwendig direkte Verbindungen zwischen Softwarekomponenten in der Top Level Composition jedes Steuergerätes zu definieren.

| SHORT-NAME | TopLevelComposition |
|---|---|
| PORTS | |
| COMPONENTS | |

COMPONENT-PROTOTYPE (9)

| | SHORT-NAME | PROVIDED-COM-SPECS | REQUIRED-COM-SPECS | TYPE-TREF |
|---|---|---|---|---|
| 1 | CentralSystemPrototype | | | /system/CentralSystemComponent |
| 2 | IoHwAbsPrototype | PROVIDED-COM-SPECS | | /IoAbstraction/IoHwAbs |
| 3 | EcuMPrototype | | | /autosar/Services/EcuM/EcuStateManager |
| 4 | DemPrototype | | | /autosar/Services/Dem/DEM |
| 5 | FensterComponentPrototype | | REQUIRED-COM-SPECS | /ID1_FensterSpiegel/FensterComponent |
| 6 | BlinkerComponentPrototype | | REQUIRED-COM-SPECS | /ID1_FensterSpiegel/BlinkerComponent |
| 7 | SpiegelComponentPrototype | | REQUIRED-COM-SPECS | /ID1_FensterSpiegel/SpiegelComponent |
| 8 | TueroeffnerComponentPrototype | | REQUIRED-COM-SPECS | /ID1_FensterSpiegel/TueroeffnerComponent |
| 9 | SpiegelheizungComponentPrototype | | REQUIRED-COM-SPECS | /ID1_FensterSpiegel/SpiegelheizungComponent |

CONNECTORS

ASSEMBLY-CONNECTOR-PROTOTYPE (10)

| | SHORT-NAME | PROVIDER-IREF | REQUESTER-IREF |
|---|---|---|---|
| 1 | AC_Fenster_Hoch_Port | PROVIDER-IREF | REQUESTER-IREF |
| 2 | AC_Fenster_Runter_Port | PROVIDER-IREF | REQUESTER-IREF |
| 3 | AC_Impulsgeber_Port | PROVIDER-IREF | REQUESTER-IREF |
| 4 | AC_Blinker_Port | PROVIDER-IREF | REQUESTER-IREF |
| 5 | AC_Spiegel_Hoch_Port | PROVIDER-IREF | REQUESTER-IREF |
| 6 | AC_Spiegel_Links_Port | PROVIDER-IREF | REQUESTER-IREF |
| 7 | AC_Spiegel_Rechts_Port | PROVIDER-IREF | REQUESTER-IREF |
| 8 | AC_Spiegel_Runter_Port | PROVIDER-IREF | REQUESTER-IREF |
| 9 | AC_Tueroeffner_Port | PROVIDER-IREF | REQUESTER-IREF |
| 10 | AC_Spiegelheizung_Port | PROVIDER-IREF | REQUESTER-IREF |

**Abbildung 4.36:** Beispielhafter Auszug aus einer Top Level Composition in Grid-Darstellung

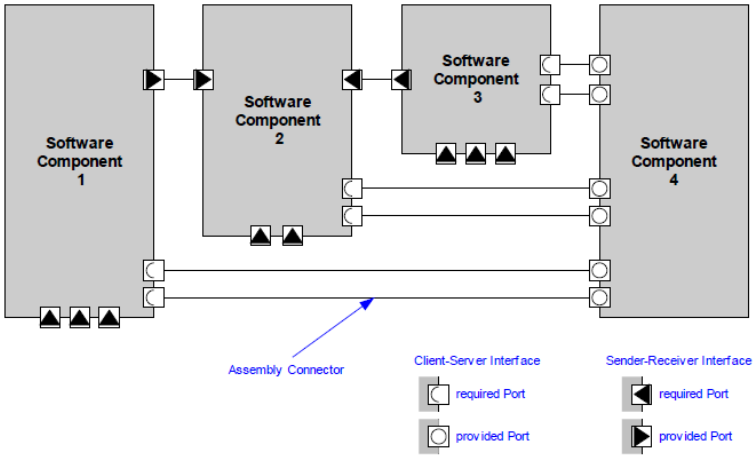


**Abbildung 4.37:** Grafische Darstellung einer Top Level Composition (prinzipielles Beispiel)

## 4.5 Konfiguration der Basis-Software

Das Software-Paket tresosECU des Herstellers Elekrobit war im Rahmen dieser Fallstudie die Kernkomponente für die Konfiguration und Kompilierung der Basis-Software. TresosECU dient zur Konfiguration der Basis-Software eines Steuergerätes und zur Anbindung von Softwarekomponenten, die manuell oder mit anderen Tools erstellt wurden. Mit tresosECU wird damit das Mapping zwischen Hardware-unabhängigen Softwarekomponenten und realen Steuergeräte-Hardware vollzogen.

Die Konfiguration der Basis-Software erfolgt über eine Auswahl an Modulen (vgl. Abbildung refseq:refIllustration6). Diese Module können einen direkten Bezug zur Steuergeräte-Hardware wie z.B. CAN-Controller oder I/O-Ports aufweisen oder einer Zwischenschicht angehören. Beim Hinzufügen des RTE-Moduls müssen alle einzubeziehenden Softwarekomponenten angegeben werden. Diese Softwarekomponenten bestehen aus Quellcode und einer Beschreibung im XML-Format. Die XML-Beschreibungen der Softwarekomponenten werden bereits in diesem Schritt ausgewertet und überprüft. Wenn zu diesem Zeitpunkt Abhängigkeiten nicht aufgelöst werden können oder andere Fehler auftreten, kann das RTE-Modul nicht eingebunden werden. Bei der Anbindung von Software-Komponten, die z.B. mit MATLAB/Simulink und TargetLink erzeugt wurden, war zum Zeitpunkt der praktischen Durchführung dieser Fallstudie noch kein durchgängiger Ablauf der Toolkette möglich, so dass es notwendig war XML-Dateien manuell zu überarbeiten und zu erstellen. Insgesamt stehen in tresosECU über 50 Module zur Verfügung, aus denen je nach Anwendungsfall die notwendigen ausgewählt werden müssen.

Die Konfiguration umfasst sowohl sehr Hardware-spezifische Einstellungen, wie z.B. das Setzen des Betriebsverhaltens von I/O-Pins oder Frequenzteilern, als auch das Anlegen und Zuweisen von Signalen für die Kommunikation. Die Modulkonfiguration ist ein sehr aufwändiger Arbeitsschritt und erfordert fundierte Kenntnisse über AUTOSAR und die Hardware des Steuergerätes. Die grafische Bedienoberfläche von tresosECU unterstützt den Konfigurationsvorgang und bietet eine Projektverwaltung. Die Modulparameter werden auf der Oberfläche strukturiert dargestellt und in vielen Fällen mit einem Menü für alle Auswahlmöglichkeiten versehen.

Die jeweiligen Parameter können von tresosECU in einem bestimmten Rahmen auf Plausibilität überprüft werden, sowohl innerhalb der Modulgrenzen als auch im Zusammenspiel mit anderen Modulen. Die konfigurierten Module eines Projektes speichert tresosECU als XML-Dateien im Projektverzeichnis für die Codegenerierung ab.

Nachdem die Konfiguration eines Steuergerätes abgeschlossen wurde, kann die Codegenerierung über die Bedienoberfläche gestartet werden. TresosECU

erzeugt aus diesen Vorgaben einen maßgeschneiderten C-Code und sogenannte Make-Dateien. Die für den Betrieb notwendigen Komponenten der AUTOSAR-Basissoftware sind ebenfalls Bestandteil des tresosECU-Paketes.

Letztendlich liegt am Ende des Anbindungs- und Konfigurationsprozesses die gesamte Betriebs-Software eines Steuergerätes im Quellcode vor und wird einem weiteren Schritt durch einen zusätzlich erforderlichen C-Compiler übersetzt.

### 4.5.1 Modulkonfiguration und Codegenerierung mit tresosECU

Die spezifische Konfiguration der AUTOSAR-Basismodule für ein Steuergerät zählt zu den elementaren Arbeitsschritten. Die Konfigurationsparameter von jedem verwendeten Modul müssen für die automatische Codegenerierung in Dateien im XML-Format vorliegen (siehe Kapitel 4.5.4).

Die Konfiguration umfasst sowohl sehr Hardware-spezifische Einstellungen, wie z.B. das Setzen des Betriebsverhaltens von I/O-Pins oder Frequenzteilern, als auch das Anlegen und Zuweisen von Signalen für die Kommunikation. Die Modulkonfiguration ist ein sehr aufwändiger Arbeitsschritt und erfordert fundierte Kenntnisse über AUTOSAR und die Hardware des Steuergerätes.

Die grafische Bedienoberfläche von tresosECU unterstützt den Konfigurationsvorgang und bietet eine Projektverwaltung. Die Modulparameter werden auf der Oberfläche strukturiert dargestellt und in vielen Fällen mit einem Menü für alle Auswahlmöglichkeiten versehen.

Die jeweiligen Parameter können von tresosECU in einem bestimmten Rahmen auf Plausibilität überprüft werden, sowohl innerhalb der Modulgrenzen als auch im Zusammenspiel mit anderen Modulen. Die konfigurierten Module eines Projektes speichert tresosECU als XML-Dateien im Projektverzeichnis für die Codegenerierung ab. Die Codegenerierung kann über der Bedienoberfläche gestartet werden. In einem Ausgabefenster erscheinen Ausgabemeldungen des Codegenerators, wie z.B.Warnungen oder Fehler.

Im folgenden Abschnitt wird die prinzipielle Vorgehensweise der Konfiguration mit tresosECU beschrieben. Das Zusammenspiel und die Abhängigkeiten zwischen einzelnen Modulen sind besonders zu beachten. Die Reihenfolge der Modulkonfigurationen ist in vielen Fällen genau einzuhalten.

#### Die Bedienungsoberfläche von tresosECU

Abbildung 4.38 zeigt die Bedienungsoberfläche von tresosECU mit einem geöffneten Projekt. Das Hauptfenster enthält in diesem Beispiel vier Fenster. Oben links befindet sich die Projekt- und Modulauswahl mit Darstellung in Baumstruktur. Darunter liegt die Parameterauswahl des aktuellen Moduls ebenfalls mit Baumstruktur. Im unteren Fenster erscheinen Statusmeldungen, Warnungen

und Fehlerausgaben. Das vierte Fenster zeigt eine Parameterseite des ausgewählten Moduls. Die Seite im Beispiel zeigt Einstellungen für ein CAN-Signal im Modul CanIf (CAN-Interface).

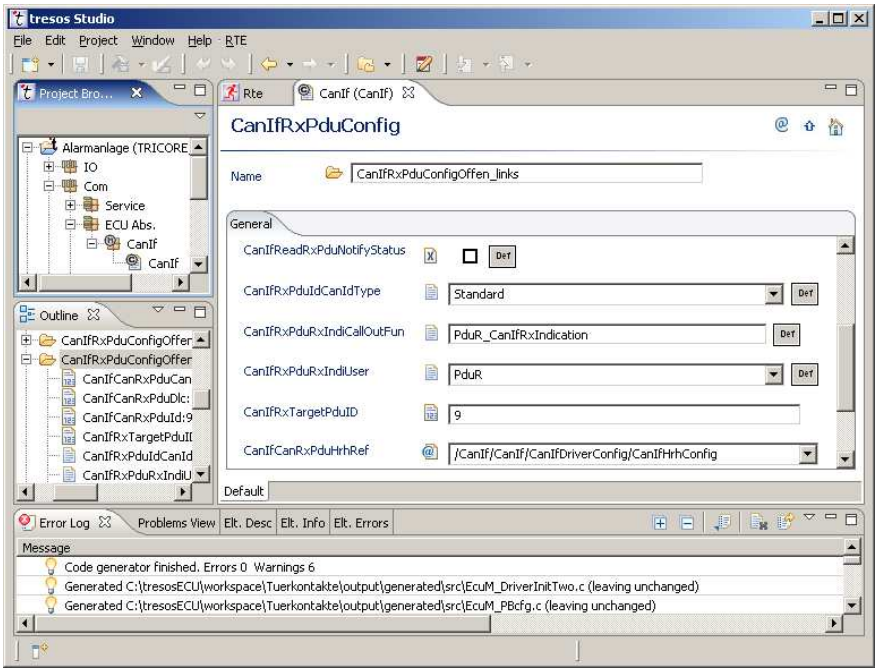


**Abbildung 4.38:** Die Bedienungsoberfläche von tresosECU

### Basiskonfiguration

In der Basiskonfiguration eines Steuergeräteprojektes wird der Typ des Mikrocontrollers und die Auswahl der verwendeten Module festgelegt. Bereits bei einfachen Projekten kann es sich dabei um 15 – 20 Module handeln. Die folgende Tabelle 4.1 zeigt die Modulauswahl für jeweils ein Steuergerät aus dem Komfortsystem.

Eine Beschreibung der Module ist im Abschnitt 2.6 zu finden. Insgesamt bietet tresosECU 38 Module für die Konfiguration an, darunter befinden sich z.B. die Module Adc (Analog Digital Converter), Icu (Input Capture Unit), MemIf (Memory Abstraction Interface), NvM (NVRAM Manager), Pwm (Pulse Width Modulation) und Spi (Serial Peripheral Interface).

Beim Hinzufügen des RTE-Moduls müssen alle einzubeziehenden Softwarekomponenten angegeben werden. Die XML-Beschreibungen der Softwarekomponenten werden bereits in diesem Schritt ausgewertet und überprüft. Wenn zu diesem Zeitpunkt Abhängigkeiten nicht aufgelöst werden können oder andere Fehler auftreten, kann das RTE-Modul nicht eingebunden werden.

| **Modul** | **Funktion** | **Layer** |
|---|---|---|
| Base | AUTOSAR Core | Base |
| Can | CAN Driver | MCAL |
| CanIf | CAN Interface | ECU Abstraction |
| Com | Communication Environment | Service |
| ComM | Communication Manager | Service |
| Dem | Diagnostic Event Manager | Service |
| Det | Development Error Tracer | Service |
| Dio | Digital Input Output Driver | MCAL |
| EcuC | ECU Configuration | Service |
| EcuM | ECU State Manager | Service |
| Gpt | General Purpose Timer Driver | MCAL |
| Make | Application Build Environment | Service |
| Mcu | Mikrocontroller Unit Driver | MCAL |
| Os | AUTOSAR Operating System | Os |
| PduR | Protocol Data Unit Router | Service |
| Port | Port Driver | MCAL |
| Rte | Runtime Environment | Runtime Environment |
| SchM | Schedule Manager | Service |

**Tabelle 4.1:** Auswahl der Module für das Komfortsystem

### Konfigurationsbeispiel für ein Nachrichtensignal

Im folgenden Beispiel wird anhand eines zu konfigurierenden CAN-Signals exemplarisch verdeutlicht, welche Abhängigkeiten der Module untereinander zu beachten und welche Schritte für ein einziges Signal durchzuführen sind. Bei dem Signal kann es sich z.B. um einen boolschen Wert handeln, der von einer Softwarekomponente von einem Steuergerät zu einem anderen geschickt werden soll. Die Arbeitsschritte müssen letztlich für alle CAN-Signale einzeln vorgenommen werden.

Die nachfolgenden Arbeitsschritte sind alle „von Hand“ auszuführen. Es werden nur jene Details dokumentiert, welche die Abhängigkeiten und die prinzipielle Vorgehensweise aufzeigen. Darüber hinaus gibt es viele Konfigurationseinträge bei denen z.B. Puffergrößen und Default-Werte zu definieren sind.

1. Im Modul EcuC muss ein Name für das Signal in der sogenannten PduCollection eintragen werden. Dies hat für einige weitere Einträge in anderen Modulen die Bedeutung einer globalen Referenz.

2. Das Signal muss im Modul PduR zu einer Liste im Bereich PduRRoutingPath mit einer Referenzierung auf den Eintrag in der PduCollection (1.) hinzugefügt werden. Für den Eintrag ist auch eine Identifikationsnummer (Source PDU ID) zu wählen, die jeweils für alle Ein- und Ausgangssignale in einer Abfolge ohne Lücken bei 0 beginnend liegen muss. Die ID 3 kann somit zweifach vorkommen, bei einem Eingangssignal und bei einem Ausgangssignal.

3. Im Modul Com ist das Signal in drei Bereichen anzulegen:

   (a) Im Bereich ComIPdu wird das Signal als Eingangs- oder Ausgangssignal definiert und ebenfalls mit einer Referenzierung auf den Namen in der PduCollection (1.) versehen. In einem weiteren Feld muss die gewählte ID aus dem PduRRoutingPath (2.) eingetragen werden.

   (b) Unter ComNetworkSignal muss das Signal mit einer Referenz auf den Eintrag unter ComIPdu (3.a) angelegt werden. Es benötigt hier ebenfalls eine Angabe zur Signalrichtung.

   (c) Das Signal ist im Bereich ComSignal anzulegen. Hier wird der Datentyp zugeordnet, Filter definiert und das Signal auf den Eintrag unter ComNetworkSignal (3.b) referenziert. Im Feld ComNotificationSignal muss bei einem Eingangssignal der Name einer Callback-Routine für die RTE angegeben werden. Fehlende Einträge an dieser Stelle sind eine häufige Fehlerursache, da in diesem Fall keine Fehlermeldungen ausgegeben werden. In der Folge erhalten Softwarekomponenten dieses Signal nicht.

4. An letzter Stelle muss das Signal im Modul CanIf definiert werden. Hier sind Referenzen auf den Eintrag unter PduCollection (1.) und die Pdu ID unter (2.) anzugeben. In diesem Modul werden auch spezifische Einträge für die CAN-Übertragung vorgenommen, wie z.B. die Festlegung der CAN-ID.

Das aufgeführte Beispiel verdeutlicht im Ansatz den enormen Aufwand für die Konfiguration eines einziges Nachrichtensignals. Viele allgemeine und Signal-spezifische Konfigurationseinträge sind zusätzlich vorzunehmen. Als einzige Hilfestellung bietet tresosECU die Möglichkeit, bestimmte Felder zu kopieren. Bei gleichen Signaleigenschaften ist dies sehr nützlich, birgt allerdings die Gefahr, bei der Nacharbeit von Hand notwendige Anpassungen von Einträgen zu übersehen.

### Konfiguration von I/O-Signalen

Zur Konfiguration von I/O-Signalen dienen die Module Dio und Port, die beide zum MCAL-Bereich gehören. Die folgenden Angaben beziehen sich ausschließlich auf den Mikrocontroller TriCore TC1766. Dieser Typ verfügt über sechs Ports mit jeweils bis zu 16 einzelnen I/O-Pins. Die dazugehörige Dokumentation des Herstellers ist unter [Inf05] zu finden.

Der Aufbau des Dio-Moduls spiegelt den Hardware-Aufbau wider. Unter den sechs Ports werden einzelne I/O-Pins oder Gruppen von I/O-Pins definiert. Die Pins werden als DioChannels und die Gruppen als DioChannelGroups bezeichnet. Über das DIO-Modul werden Signalnamen Pins oder Pin-Gruppen zugeordnet. Unter einem Port können einzelne Pins zu einer Gruppe zusammengefasst werden, um z.B. mit nur einem Signalnamen auf die acht Bits eines Busses unter Berücksichtigung ihrer Wertigkeit zugreifen zu können.

Auch das Port-Modul ist direkt auf die spezielle Hardware des jeweiligen Mikrocontrollers bezogen. Bei der Konfiguration werden die I/O-Eigenschaften für einzelne Pins vorgenommen. Dazu zählen beispielsweise die Signalrichtung, die Polarität oder der Treibertyp. Die Pins haben eine ID, die beim TC1766 zwischen 0 und 95 liegt. Beim DIO-Modul bezieht sich die Pin-ID immer auf den dazugehörigen Port, d.h. es gibt die ID 3 sechsfach. Die Zuordnung eines DioChannels auf einen Pin des Port-Moduls muss über das Datenblatt von Hand aufgelöst werden.

Über die Namen der DioChannels wird aus der I/O Hardware Abstraction-Komponente auf die I/O-Pins zugegriffen. Die Namen müssen aus diesem Grund auch in die C-Datei der Komponente geschrieben werden. Das Prinzip wird in Kapitel 4.5.2 erläutert.

### Konfiguration der RTE

Die Konfiguration des RTE-Moduls erfolgt erst nachdem alle benötigten Nachrichtensignale in den darunter liegenden Schichten definiert wurden, da sie refe-

renziert werden müssen. Im Wesentlichen erfolgt bei der RTE-Konfiguration das Zuordnen von Runnables aus den angebundenen Softwarekomponenten auf Tasks vom AUTOSAR OS und das Zuordnen von Nachrichtensignalen auf Signale, die im COM-Modul definiert wurden.

Eine Runnable wird im Normalfall dem RteTask1, dem RteTask2 oder dem InitTask zugewiesen. Der InitTask wird einmalig beim Systemstart ausgeführt, d.h. eine zugewiesene Runnable wird auch genau zu diesem Zeitpunkt aufgerufen. In Orientierung an Beispielprojekten des Herstellers erfolgte die Zuordnung der Runnables bei den Steuergeräten des Komfortsystems nach einem einfachen Schema. Alle Runnables, die periodisch gestartet werden, liegen auf RteTask1, während alle durch Eintreffen von Nachrichten zu startenden Runnables RteTask2 zugeordnet sind.

Im Bereich S/R Data Mapping (S/R = Sender/Receiver) findet eine Anbindung einzelner Nachrichtensignale der Softwarekomponenten an COM-Signale statt. Die Bedienungsoberfläche unterstützt die Zuordnung, indem sie aus der gesamten Auswahl nur COM-Signale anbietet, die von Datentyp und -richtung geeignet sind. Ein Fehler in der vorliegenden Version verhindert jedoch das Zuweisen von Signalen vom Typ Array. Für diese Signale musste ein Workaround gewählt werden, beim dem die Signale „von Hand“ in den XML-Dateien verbunden wurden.

### Konfiguration des OS-Moduls

Im OS-Modul werden u.a. Interrupt Service Routinen eingetragen. Für die Steuergeräte des Komfortsystem war es erforderlich eine Eintragung für den Interrupt, der beim Senden von Daten über die serielle Schnittstelle ausgelöst wird, im Bereich OsIsr vorzunehmen. Das Ausgeben von Daten über die serielle Schnittstelle dient zur Ausgabe von Debug-Informationen.

### Anmerkungen zur Modulkonfiguration

Die Konfiguration der Module erfordert umfangreiche Detailkenntnisse über die Steuergeräte-Hardware und über die AUTOSAR-Basis-Software. Das Anlegen einer vollständig neuen Konfiguration für ein Steuergerät ist ohne längere Einarbeitungszeit nicht zu bewältigen. Ein praktikabler Weg ist nach den Erfahrungen aus der Fallstudie, eine bestehende Konfiguration umzuarbeiten. Im konkreten Fall wurde als Grundlage für jedes Steuergerät des Komfortsystems die vorliegende Konfiguration eines Beispielprojektes ausgewählt. Dort waren unzählige Einträge, die ohne Änderung übernommen werden konnten, bereits auf sinnvolle Werte gesetzt. Dazu zählen beispielsweise Einstellungen für die System-Timer und das Betriebssystem. Bei den Nachrichtensignalen konnten bestehende Signa-

le umbenannt und kopiert werden, so dass deren Detaileinstellungen nicht neu angelegt werden mussten.

### 4.5.2 Besonderheiten der I/O Hardware Abstraction-Implementierung

Im Allgemeinen zählen Ein- und Ausgabezugriffe auf die Port-Pins eines Mikrocontrollers zu den elementaren Abläufen innerhalb eines Steuergerätes. Die entsprechende Anbindung an die AUTOSAR-Basis-Software fällt in den Bereich der I/O Hardware Abstraction. Die Spezifikation von AUTOSAR gibt für die Implementierung der I/O Hardware Abstraction innerhalb der Basis-Software nur eine Richtlinie vor. Im Kapitel 2.6 in den Abschnitten *I/O Hardware Abstraction* und *Richtlinie zur Impl. der I/O Hardware Abstraction* sind die Konzeption und die Richtlinie beschrieben. Die Spezifikation selbst findet sich in [AUT07k].

Die Anbindung und die Implementierung der I/O Hardware Abstraction in tresosECU in der Version 2007.b werden im folgenden Abschnitt erläutert. Sie weichen maßgeblich von den Konzepten ab, die für andere Bereiche der Basis-Software gelten. Für jedes Steuergerät ist eine spezifische Softwarekomponente zu erstellen, die für jeden eingeplanten I/O-Port einen eigenen Anschluss nach dem Client/Server-Prinzip aufweisen muss. Diese Softwarekomponente wird im weiteren als I/O Hardware Abstraction-Komponente bezeichnet. Ihr konkreter Aufbau besteht aus einer XML-Datei und einer C-Datei. Abbildung 4.39 zeigt ein Beispiel für den Aufbau eines Steuergerätes mit einer I/O Hardware Abstraction-Komponente in einer schematischen Darstellung.

Die Applikationsebene enthält drei Softwarekomponenten und die spezifische I/O Hardware Abstraction-Komponente, die in der Top Level Composition ebenfalls als Softwarekomponente eingebunden wird.

Die Anschlüsse für I/O-Zugriffe von allen Applikations-Softwarekomponenten eines Steuergerätes (z.B. Nr. 1-3 in Abb. 4.39) werden mit den entsprechenden Anschlüssen der I/O Hardware Abstraction-Komponente verbunden. Alle Ports für I/O-Zugriffe auf Seiten der Applikations-Softwarekomponenten sind vom Typ Required (Anschluss erforderlich), die Gegenstücke der I/O Hardware Abstraction-Komponente sind vom Typ Provided (Anschluss bereitgestellt). Die Kommunikations-Ports der Applikations-Softwarekomponenten werden mit der RTE verbunden. Die Verbindungen zwischen den I/O-Anschlüssen sind in diesem Beispiel waagerecht eingezeichnet. Die Verbindungen der Anschlüsse für die Kommunikation sind senkrecht und gestrichelt dargestellt.

Im unteren Bereich der Abbildung befinden sich die Schichten der Basis-Software. Um die Unterschiede und Parallelen der Implementierung des I/O- und des Kommunikationsbereiches zu verdeutlichen, sind auch die verschiedenen

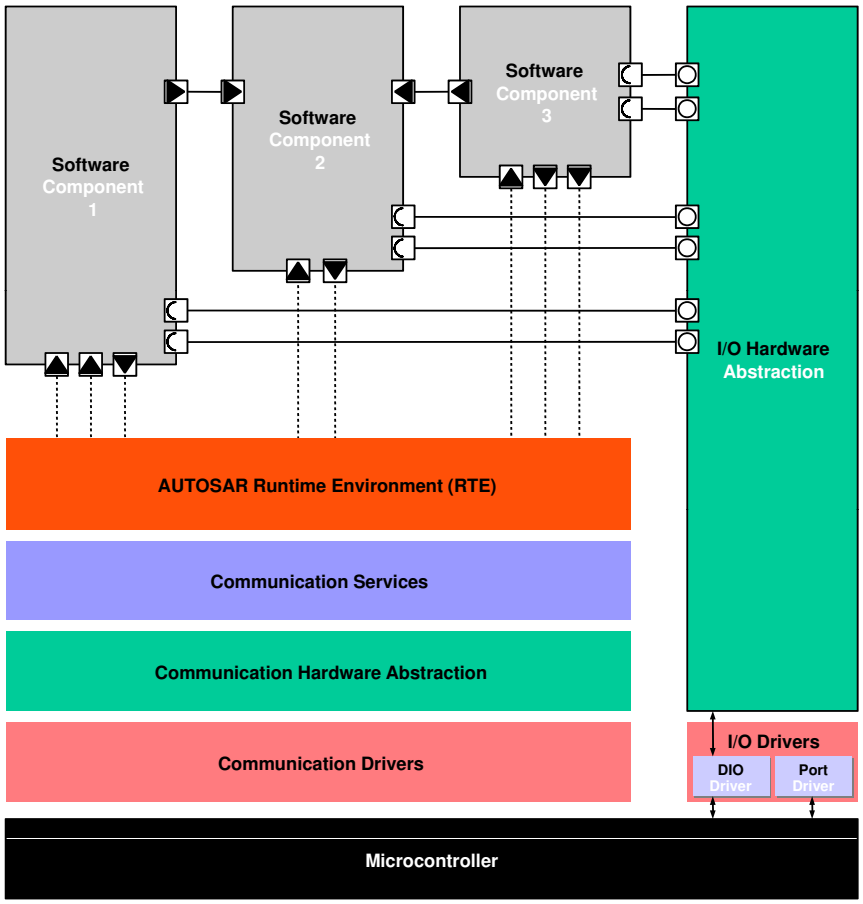


**Abbildung 4.39:** Anbindung der I/O Hardware Abstraction als Softwarekomponente

Schichten für die Kommunikation dargestellt. Eine Dokumentation des Kommunikationsbereiches sowie Literaturverweise sind im Abschnitt 2.6 zu finden.

Die I/O Hardware Abstraction-Komponente greift auf Module der I/O-Treibeschicht zu. Dargestellt sind lediglich die Module DIO Driver und Port Driver. Die I/O-Treiberschicht stellt weitere Module zur Verfügung, die im Kapitel 2.6, Abschnitt *I/O Hardware Abstraction* beschrieben sind. Im Rahmen der Fallstudie wurden nur die zwei angegebenen Module verwendet. In der folgenden Beschreibung werden weitere Treibermodule nicht berücksichtigt.

Die Definition der bereitgestellten Anschlüsse erfolgt für die I/O Hardware Abstraction-Komponente in einer XML-Datei nach dem Schema, das allgemein für Softwarekomponenten gilt. Jeder dort definierte Anschluss besitzt einen eindeutigen Namen, mehrere Referenzeinträge und einen Zahlenwert mit Indexfunktion. Dem Index kommt eine besondere Rolle zu. Alle Ein- und Ausgänge werden durchnummeriert, d.h. der Index für jeweils alle Eingangs-Ports und alle Ausgangs-Ports wird bei 0 beginnend hochgezählt.

Im Abschnitt 4.4.7 sind die entsprechenden Teile als XML-Definition dargestellt. Neben einer XML-Datei ist auch Programm-Code für die I/O Hardware Abstraction-Komponente erforderlich, der in eine C-Datei geschrieben wird. Der C-Code besteht im wesentlichen aus den Definitionen von Arrays für jeweils alle Eingangs- und alle Ausgangs-Ports sowie zwei Funktionsdefinitionen. Es gibt jeweils eine Funktion für alle Port-Ausgaben und eine für alle Port-Eingaben. Der folgende C-Code zeigt exemplarisch den Aufbau von Array und Funktion für die Port-Ausgabe:

```
static const Dio_ChannelType OutputChannels[3] =
  {
    Error_LED,    /* Index 0 */
    Busy_LED,     /* Index 1 */
    Debug_LED     /* Index 2 */
  };
Std_ReturnType IoHwAb_SetDiscrete(UInt8 channel, IoHwAb_BoolType State)
{   Std_ReturnType ret = RTE_E_DiscreteOutputIf_E_NOT_OK;
  if (channel < 3)
  {
  Dio_WriteChannel(OutputChannels[channel], (Dio_LevelType) State);
  ret = E_OK;
  }
  return ret;
}
```

Der Aufbau von Array und Funktion für die Port-Eingabe erfolgt analog. Jeder verwendete Eingabe- und Ausgabe-Port muss im tresosECU-Projekt als soge-

nannter DIO Channel im Modul DIO konfiguriert werden. Die dort eingetragenen Namen müssen denen im jeweiligen Array entsprechen. Alle Eingangs-Ports bilden ein Array ebenso wie alle Ausgangs-Ports. Von entscheidender Bedeutung ist die Reihenfolge der Einträge. Jeder Port-Name muss im Array genau an jener Stelle stehen, die seinem Index in der XML-Datei entspricht. Die Zählung beginnt auch im Array bei 0. Die angegebene Funktion IoHwAb_SetDiscrete() hat als Parameter den Index und den gewünschten Ausgabezustand des Port-Pins. Sie übergibt lediglich den DIO Channel aus dem Array und den Ausgabezustand an die Funktion Dio_WriteChannel().

Neben der Konfiguration der DIO Channels ist auch die Konfiguration der Port-Pins im tresos-ECU-Projekt erforderlich. Mit ihr werden für jeden verwendeten Port-Pin die Betriebsparameter festgelegt, z.B. ob der Pin für Eingabe- oder Ausgabe genutzt werden soll.

Diese Lösung für die I/O Hardware Abstraction ist sehr aufwendig in der Erstellung und Wartung. Es ist ein hohes Maß an „Handarbeit“ erforderlich. Die Vergabe der Indizes muss parallel in zwei Dateien vorgenommen werden. Ein einziger Fehler kann Auswirkungen auf eine große Zahl von Pins haben.

### 4.5.3 Projektverwaltung und Verzeichnisstruktur

Bei der Durchführung der Entwicklungsarbeiten war das Erstellen, Bearbeiten oder Überprüfen einzelner Dateien des Projektes erforderlich. Beispielsweise müssen Makefiles beim Hinzufügen oder Entfernen von Quelldateien mit einem Texteditor überarbeitet werden. Dateien, die die Konfiguration von Softwarekomponenten enthalten, müssen teilweise neu erstellt oder modifiziert werden. In bestimmten Fällen ist es sinnvoll, automatisch generierte Quelldateien zu überprüfen oder zu debuggen. Die Projektverwaltung auf Applikations- und Gesamtebene erfolgt im Prinzip manuell.

In den folgenden Abschnitten wird der Aufbau der Verzeichnisstruktur eines Projektes, das Prinzip für die automatische Generierung von Quellcode und die Grundlagen für die Compilierungsphase der Projekte erläutert. Alle Angaben beziehen sich dabei auf die Projektumgebung tresos-ECU von Elektrobit.

Das Projektmanagement von tresosECU erlaubt es, mehrere Projekte gleichzeitig zu verwalten. Die Projekte befinden sich in einem sogenannten workspace (dt.: Arbeitsbereich), einem Ordner im Dateisystem, der sich an beliebiger Stelle befinden kann. Es ist möglich, zwischen mehreren Arbeitsbereichen zu wechseln. Die Projekte sind Unterordner im workspace und enthalten selbst einige Verzeichnisse. Projektordner enthalten die Konfigurationsdateien und den Quellcode für jeweils ein Steuergerät. Die komplette Verzeichnisstruktur eines Beispielprojektes ist in Abbildung 4.40 dargestellt.

Aus der Vielzahl aller Unterordner werden diejenigen kurz beschrieben, die erfahrungsgemäß häufig geöffnete Dateien enthalten. Im Verzeichnis config befinden sich die Konfigurationsdateien des Steuergerätes im XML-Format. Hier liegt die XML-Beschreibung der AUTOSAR-Softwarekomponenten, die Compositions-Datei und mittels tresosECU konfigurierte Module der Basis-Software. Die Konfigurationsparameter der Basismodule und der RTE werden in Dateien mit der Endung .xdm gespeichert. Die Parameter der MCAL-Module liegen in Abhängigkeit vom verwendeten Mikrocontroller in entsprechend benannten Unterordnern (im Beispiel TRICORE/TC1766). Einige Dateien, die von Hand erstellt oder bearbeitet werden müssen, liegen im config-Verzeichnis.

Die Ordner doc und simulink sind nicht in der üblichen Verzeichnisstruktur von tresosECU-Projekten enthalten. Sie wurden im Rahmen der Fallstudie bei allen Projekten angelegt, um alle weiteren für die Entwicklung relevanten Dateien einzugliedern. Der Ordner doc ist für Projektspezifikationen gedacht, die das jeweilige Steuergerät betreffen. Im Ordner simulink liegen die Projektdateien für die MATLAB/Simulink-Modelle inklusive der von TargetLink erzeugten Quelldateien. Im Ordner output werden in der Generierungs- und Compilierungsphase maschinell erzeugte Dateien abgelegt. Im dortigen Unterordner bin befinden sich nach erfolgreichem Compilierungsvorgang die Binärdateien zum Flashen und Debuggen des Steuergerätes. Im Unterordner generated werden in der Generierungsphase alle automatisch erzeugten Konfigurations- und Quelldateien abgespeichert. Generierte C-Quelldateien befinden sich in den Ordnern include und src und den jeweils tiefer geschachtelten Ordnern TRICORE/TC1766. Bei bestimmten Problemen kann es sinnvoll sein, rekursiv alle Dateien im Ordner output zu löschen. Bei einem solchen Löschvorgang müssen alle Dateien im Unterordner swcd erhalten bleiben, da sie in der Generierungsphase benötigt werden.

Alle nicht von tresosECU automatisch generierten C-Quelldateien sollten im Ordner source/application liegen. Zum Basisumfang eines Projektes gehören einige C- und Header-Dateien, die sich in diesem Ordner befinden. Selbst entwickelter oder von TargetLink erzeugter C-Code sollte in diesem Ordner liegen, ebenso wie der Quellcode der angepassten I/O Hardware Abstraction-Komponente.

Der Ordner util enthält Make-Dateien, Konfigurationsdateien für die Compilierung und Batch-Dateien. Der Compilierungslauf läuft in einer Shell (Kommandozeile / Eingabeaufforderung) ab. Vor dem ersten Lauf müssen Umgebungsvariablen gesetzt werden. Dazu wird im Ordner util die Batch-Datei launch.bat ausgeführt. In die Datei Makefile.mak müssen eigene C-Quelldateien eingetragen werden, damit sie compiliert und eingebunden werden können.

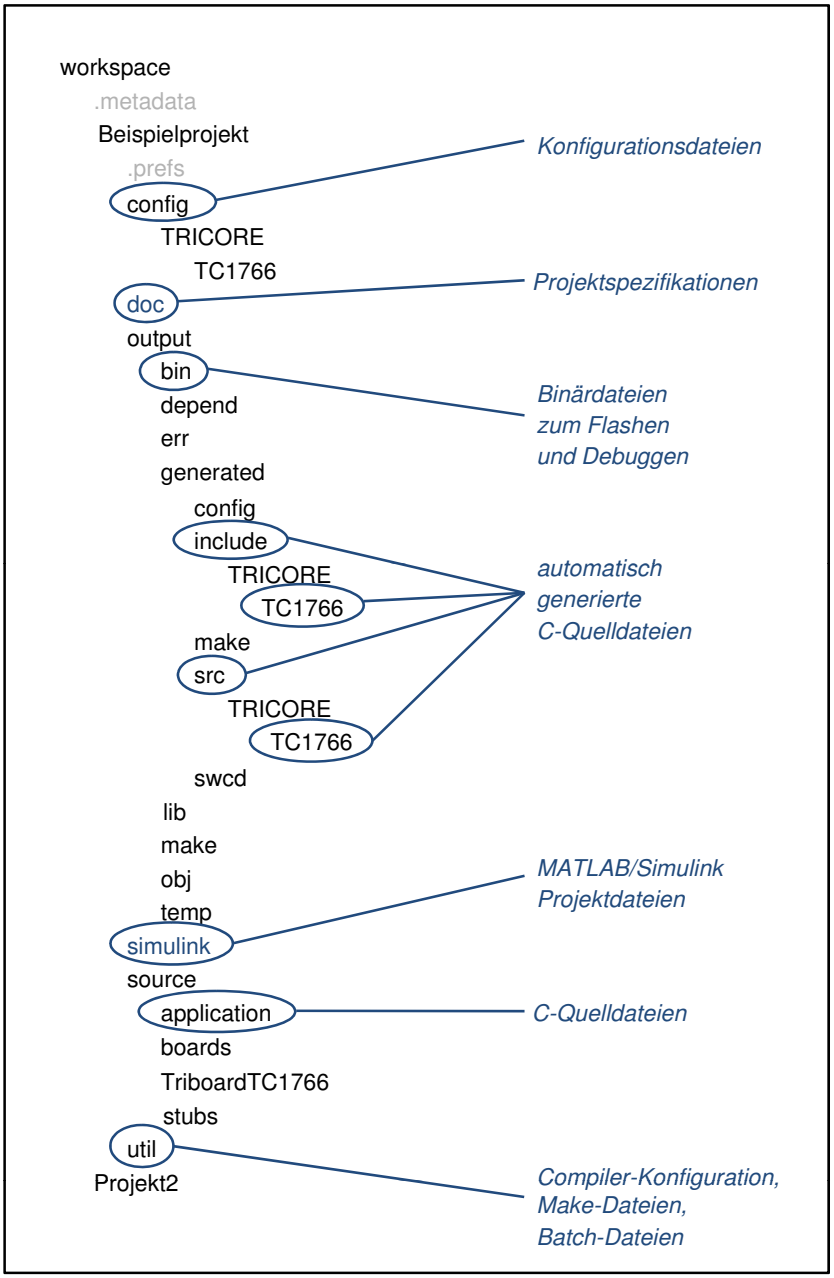


**Abbildung 4.40:** Verzeichnisstruktur von Projekten mit tresosECU

### 4.5.4 Codegenerierung mit tresosECU

Nachdem die Software-Module eines Steuergerätes mit der Bedienungsoberfläche von tresos-ECU vollständig konfiguriert wurden, wird in der Generierungsphase mit den Konfigurationsparametern C-Quellcode für die einzelnen Module maschinell erzeugt. Der Codegenerator liest Vorlagen der Module ein, die aus C-Quelltext mit speziellen Makros bestehen. Die verschiedenen Makros werden unter Einbeziehung der dazugehörigen Konfigurationsdaten in neuen C-Quellcode umgewandelt.

Für jedes Modul eines Projektes legt tresosECU eine XDM-Datei im Unterordner config an (s. Abb. 4.40). Die XDM-Datei enthält konfigurierbare Parameter des Moduls mit ihren hierarchischen Abhängigkeiten. Das XDM-Format (XDM = XPath Data Model, siehe [Wor07b]) ist ein spezielles XML-Dokumentenformat, welches unter AUTOSAR u.a. zum Speichern von Datenstrukturen genutzt wird. Das zugrunde liegende Konzept wird als Data Model bezeichnet. Das Data Model besteht aus beliebig vielen Knotenpunkten, die hierarchisch in einer Baumstruktur organisiert werden. Die Baumstruktur wird grundsätzlich zweifach repräsentiert, in einem Schema Tree und einem Data Tree. Abbildung 4.41 zeigt beispielhaft den einfachsten Fall, in dem Schema Tree und Data Tree die gesamte Struktur der Knoten in paralleler Ausführung enthalten. Die beiden Bäume unterscheiden sich prinzipiell in den Knotentypen. Der Schema Tree beschreibt die Strukturen und der Data Tree enthält die jeweiligen Konfigurationsdaten. Jedes Knotenelement im Datenbaum besitzt eine Referenz auf die Strukturbeschreibung im entsprechenden Knoten des Schemabaumes. Den Knoten können Attribute mit entsprechenden Werten zugeordnet werden.

In der Generierungsphase werden für jedes Modul die XDM-Dateien eingelesen und ausgewertet. In den Makros der Code-Vorlagen werden XPath-Ausdrücke verwendet, mit denen die Konfigurationsparameter bestimmter Knoten adressiert, Werte für Attribute ausgelesen und in C-Code umgewandelt werden können. XPath steht für XML Path Language und ist eine Abfragesprache mit der Teile von XML-Dokumenten adressiert werden können. Eine Definition von Xpath 2.0 ist unter [Wor07a] zu finden.

Folgendes Beispiel zeigt einen Ausschnitt aus einer Code-Vorlage und den vom Generator daraus erzeugten C-Code:

```
switch( CbkType )
{[!/*
*/!][!IF "$ulCanTpUsed"!]
case 1:
[!"CanIfInitConfig/*/CanIfRxPduConfig/*[CanIfRxPduRxIndiUser = 'CanTp']
[1]/CanIfRxPduRxIndiCallOutFun"!]( TargetPduId, &CanTpPduInfo );
break;[!/*
```

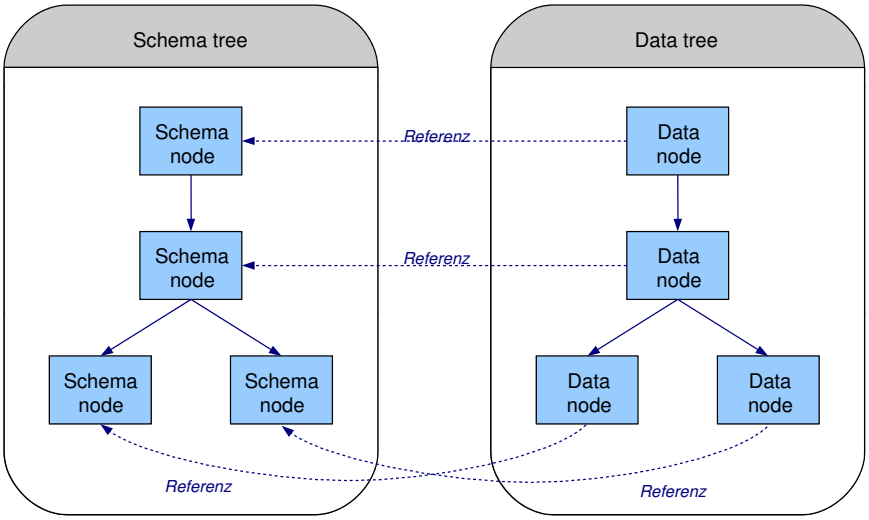


**Abbildung 4.41:** Data Model mit Schema Tree und Data Tree

```
*/![!ENDIF!][!/*
*/![!IF "$ulPduRUsed"!]
case 2:
[!"CanIfInitConfig/*/CanIfRxPduConfig/*[CanIfRxPduRxIndiUser = 'PduR']
[1]/CanIfRxPduRxIndiCallOutFun"!]( TargetPduId, CanSduPtr );
  break;[!/*
*/![!ENDIF!]
default:
/* Erroneous configuration or no RX indication shall be transmitted */
break;
}
```

Durch die Konfigurationsparameter ergibt sich daraus folgender C-Code:

```
switch( CbkType )
{
case 2:
PduR_CanIfRxIndication( TargetPduId, CanSduPtr );
break;
default:
/* Erroneous configuration or no RX indication shall be transmitted */
break;
}
```

Im Beispiel ist zu erkennen, dass komplette Code-Segmente ein- oder ausgeschaltet werden können. Der Abschnitt „case 1:“ ist im Code nicht mehr vorhanden. Eine andere Möglichkeit besteht darin, Zeichenketten, Zahlenwerte oder weitere Datentypen durch XPath-Ausdrücke erzeugen zu lassen. Aus einem XPath-Ausdruck wurde im Beispiel-Code der Funktionsname „PduR_CanIfRxIndication“ generiert.

Wie in AUTOSAR vorgesehen, ist es möglich, eigene konfigurierbare Module zu erstellen und in tresosECU einzubinden. Die Vorgehensweise ist in [Ele07] dokumentiert. Dort wird auch das zugrunde liegende Data Model und die Nutzung von XPath beschrieben.

## 4.6 Testen

Im Rahmen dieser Fallstudie wurde nach erfolgreicher Implementierung der Funktionalität eine Überprüfung der definierten Anforderungen auf Basis eines automatisierten Testsystems durchgeführt.

Das Testen wird prinzipiell sowohl als notwendiger Schritt zur Steigerung der Qualität als auch zur Überprüfung der geforderten Qualität verstanden. Wie bereits in Abschnitt 3.4 erläutert, ist die Überprüfung der Korrektheit der umgesetzten formalen Anforderungen bereits in einer frühen Phase wichtig. Dabei wird Korrektheit als Qualitätsmerkmal verstanden. Nach [Lig02] kann Korrektheit nicht graduell abgestuft werden. Software ist korrekt, wenn sie fehlerfrei und konsistent zu ihrer Spezifikation ist. Die Erfüllung dieses Anspruchs wird durch eine auf dem AUTOSAR-Standard basierende Methodik unterstützt, das Ergebnis hiervon ist schließlich durch Tests zu überprüfen. Dabei stellt die Methodik einen zentralen Punkt dar, da nach DIN EN 50128 allein durch Testen die Korrektheit nicht nachgewiesen werden kann.

*„Program testing can be used to show the presence of bugs, but never to show their absence.“* (E. Dijkstra)

Durch Tests können allgemein in einem System Fehler entdeckt, lokalisiert und letztlich korrigiert werden. Weiterhin ermöglichen Tests die Beurteilung der Leistungsfähigkeit des Systems anhand definierter Qualitätsmerkmale [Lig05].

Nach [Mye89] ist *„Testen [..] der Prozeß, ein Programm mit der Absicht auszuführen, Fehler zu finden.“*. Im V-Modell ist das Testen ein integraler Bestandteil des Entwicklungsprozess. Bereits in frühen Phasen der Entwicklung, beispielsweise als Ableitung aus der Anforderungsdefinition für das Gesamtsystem, werden Testfälle für den Systemtest definiert. Durch Testfälle werden die einzelnen durchzuführenden Tests formuliert, mit dem Ziel, durch möglichst wenige Tests möglichst viele Fehler zu finden, um eine möglichst hohe Testqualität zu erreichen [Lig05].

Allgemein wird der Begriff *„Fehler“* für unterschiedliche Sachverhalte verwendet. Daher soll im Folgenden kurz zwischen verschiedenen Begrifflichkeiten in Anlehnung an *„IEEE Standard Glossary of Software Engineering Terminology“* [IEE90] unterschieden werden.

**Versagen** Ein Versagen (failure) eines Systems oder einer Komponente ist die Unfähigkeit desselben, die geforderte Funktion bzw. Anforderung innerhalb definierter Grenzen zu erfüllen.

**Fehlfunktion** Eine Fehlfunktion (fault) stellt einen Defekt einer Hardwarekomponente bzw. einen inkorrekten Schritt oder Prozess eines Softwaresystems dar.

**Fehler** Als Fehler (error) wird die Abweichung zwischen einer berechneten, beobachteten oder gemessenen Größe und der wahren, spezifizierten oder theoretisch korrekten Größe verstanden. Darüber hinaus kann ein Fehler eine Handlung eines Benutzers sein, die ein inkorrektes Ergebnis hervorruft.

### 4.6.1 Überblick Testmethodik

Eine zentrale Rolle des Testens stellt sowohl die Testmethodik zur Durchführung der Tests als auch die Auswahl geeigneter Testfälle dar. Die Vorgehensweise lässt sich durch verschiedene Kriterien klassifizieren. Abbildung 4.42 zeigt eine mögliche Klassifikation von Software-Prüftechniken. Diese grafische Übersicht erhebt dabei keinen Anspruch auf Vollständigkeit. Ein wesentliches Merkmal ist die Ausführung des Programmcodes. Durch statische Techniken wird der zu testende Programmcode verifiziert bzw. analysiert ohne eine eigentliche Ausführung der Software. Darunter fällt auch die bei sicherheitskritischen Anwendungen zunehmend wichtiger werdende formale Verifikation, durch die auf Basis von meist mathematischen Modellen die Korrektheit der Software bezüglich ihrer Spezifikation nachgewiesen werden kann, das wiederum das Vorliegen einer formalen Spezifikation bedingt.

Dynamische Verfahren hingegen sind dadurch gekennzeichnet, dass der Programmcode während des Testens ausgeführt wird. Innerhalb der dynamischen Verfahren finden sich insbesondere strukturorientierte Verfahren sowie funktionsorientierte Verfahren.

Strukturorientierte Tests leiten sich aus der Struktur der Software ab. Meist basieren diese Tests auf dem Kontrollfluss bzw. dem Datenfluss des zu testenden Systems. Durch die Bestimmung des Kontroll- oder Datenflusses ist es möglich, Überdeckungskriterien zu definieren, um ein Maß für die Vollständigkeit der Tests zu gewinnen. In der Praxis relevant sind besonders die Zweigüberdeckung und Bedingungsüberdeckungen. Eine vollständige Pfadüberdeckung eines komplexeren Systems lässt sich aufgrund der immensen Zahl der möglichen Pfade hingegen kaum erreichen.

Funktionsorientierte Tests beruhen auf der Spezifikation des Systems und weisen in der Regel nur *„Stichprobencharakter“* auf, da ein Test aller möglichen Eingaben nur in trivialen Fällen erreichbar ist. Daher wird versucht, mit möglichst wenigen Testfällen möglichst viele Fehler zu entdecken. Anhand der Spezifikation können durch verschiedene Methoden systematisch Testfälle definiert werden. Die Methode der funktionalen Äquivalenzklassenbildung versucht durch gezielte Bildung von Äquivalenzklassen repräsentative Vertreter gleichartiger Eingabedaten zu finden und daraus entsprechende Testfälle abzuleiten. Basiert die Spezifikation auf Zustandsautomaten, die ein in der Praxis weit verbreitetes Mittel zur Beschreibung von Systemen darstellen, können daraus zustandsbasierte Tests entworfen werden. Beim zustandsbasierten Test wird versucht jeden spe-

zifizierten Zustand bzw. Zustandsübergang zu überprüfen, wobei auch gerade die nicht erlaubten Zustandsübergänge zu betrachten sind. Generell können bei ausschließlich funktionsorientierten Tests allerdings noch Programmteile bisher ungetestet verbleiben.

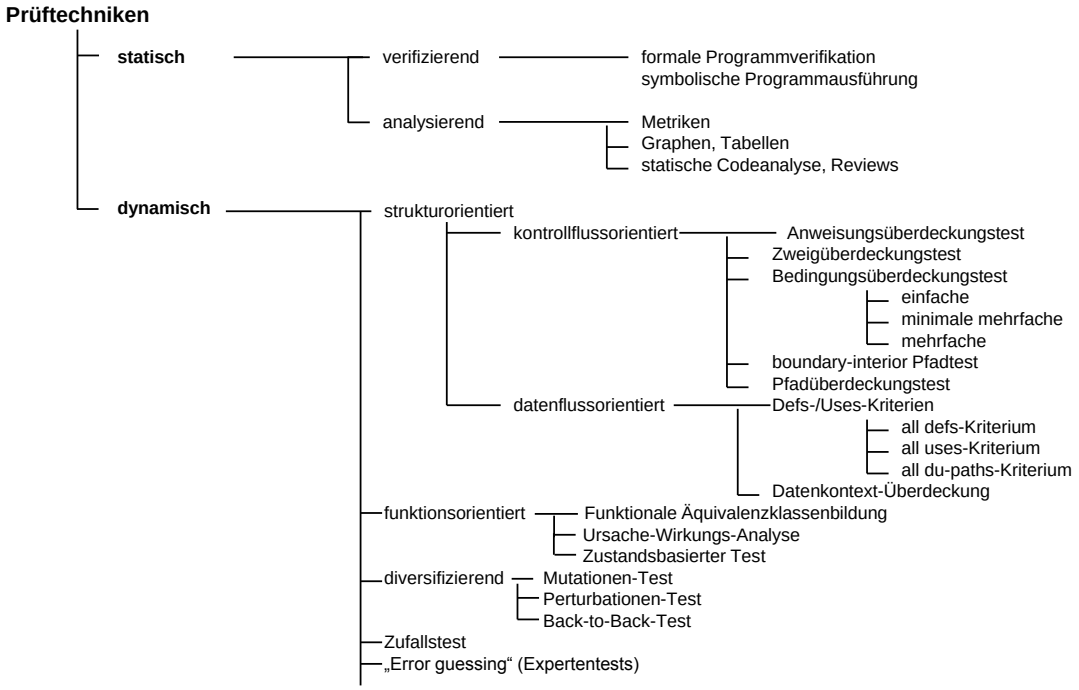


**Abbildung 4.42:** Klassifikation der Software-Prüftechniken nach [Lig02] (Auswahl)

Weiterhin lassen sich die dynamischen Tests bzw. Prüftechniken durch ihre Sichtweise auf das System einordnen. Die strukturorientierten Tests können auch als sogenannte White-Box-Tests bezeichnet werden, da die innere Struktur zur Testfallerzeugung analysiert wird. Die Wahl der Testfälle resultiert meist in einem systematischen Durchlaufen der Software. Durch die Bindung an die Struktur der Software werden entsprechend der Testmethodik auch implementierte Programmteile geprüft, die vorher nicht spezifiziert wurden.

Im Gegensatz zu den White-Box-Tests sind bei den sogenannten Black-Box-Tests die internen Abläufe und Zustände der Software unbekannt bzw. irrelevant für die durchzuführenden Tests, da die Testfälle nur aus der Spezifikation ermittelt werden. Dementsprechend lassen sich alle funktionsorientierten Tests den Black-Box-Tests zuordnen. Durch funktionsorientierte Testmethoden können fehlende Programmteile gefunden werden, die zwar spezifiziert, aber nicht imple-

mentiert worden sind. Umgekehrt sind jedoch nicht alle Black-Box-Tests auch funktionsorientiert.

Nicht funktionsorientierte Black-Box-Tests stellen beispielsweise die Zufallstests dar. Die Eingabedaten der Zufallstests werden zufällig generiert. Dabei wird die gezielte und systematische Auswahl von Testfällen durch eine große Anzahl zufälliger Testfälle ersetzt. Diese zufallsbasierten Tests benötigen zwangsläufig eine automatisierte Auswertung, um die hohe Zahl der generierten Testfälle auswerten zu können. Erst dadurch steigern Zufallstests auch deutlich die Effizienz [SMWM09].

Für die Automatisierung der Auswertung ist ein sogenanntes Testorakel notwendig, um für einen durchgeführten Testfall die Entscheidung treffen zu können, ob das Verhalten des jeweiligen Moduls oder Systems fehlerhaft ist [Leh04]. Das Testorakel muss dementsprechend in Abhängigkeit der Eingangsdaten Kenntnis über das korrekte Ausgangsverhalten besitzen.

### 4.6.2 Testen im V-Modell

Wie in Abbildung 4.43 dargestellt, findet sich das Testen in allen Stufen im rechten aufsteigenden Ast des klassischen V-Modells. In der Fallstudie wurde, wie bereits in Abschnitt 3.3 beschrieben, entsprechend eines modifizierten V-Modells entwickelt.

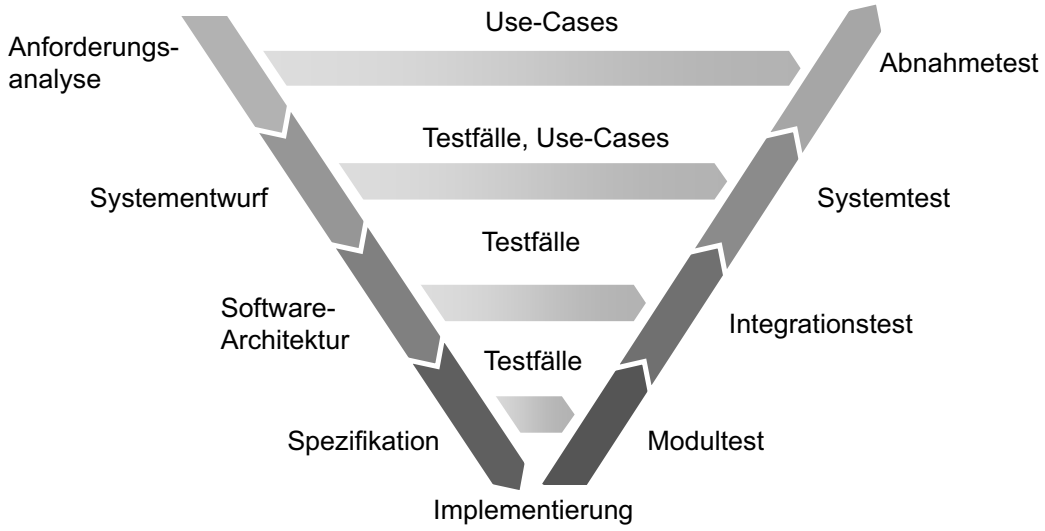


**Abbildung 4.43:** Klassisches V-Modell nach [Hof08]

Das Testen ist nach dem klassischen V-Modell in die folgenden vier Stufen gegliedert:

**Modultest** Der Modultest hat das Ziel jedes Modul bzw. auch nur Teile eines Moduls bis hin zu einzelnen Klassen isoliert zu testen. Dabei bilden sogenannte Testtreiber den Rahmen für den Modultest, indem diese die aufrufenden Module ersetzen. Dadurch bleibt auch bei umfangreichen Systemen die Komplexität beherrschbar. In der Regel werden zum Modultest überwiegend strukturorientierte Testverfahren eingesetzt.

**Integrationstest** Die nächsthöhere Abstraktionsstufe zum Modultest bildet der Integrationstest. Dabei werden die Module schrittweise zu einem Gesamtsystem verbunden. Der Integrationstest soll zeigen, dass die einzeln getesteten Module wiederum ein funktionsfähiges Gesamtsystem bilden [Hof08]. Für den Integrationsprozess können neben den Black-Box-Tests auch noch teilweise White-Box-Testverfahren eingesetzt werden.

**Systemtest** Vom vollständigen Integrationstest besteht ein fließender Übergang zum Systemtest. Allerdings kommen beim Systemtest größtenteils nur noch funktionsorientierte Testverfahren zum Einsatz, da die Überprüfung der spezifizierten Funktionalität im Vordergrund steht. Die innere Struktur des Systems wird demnach mit steigendem Abstraktionsgrad zunehmend irrelevant.

**Abnahmetest** Der Abnahmetest stellt eine spezielle Art des Testens dar. Beim Abnahmetest wird wie beim Systemtest das System gegen die Spezifikation getestet. Das Ziel im Abnahmetest ist jedoch nicht mehr, die Qualität zu steigern und Fehler zu finden, sondern nur noch zu demonstrieren, dass das System die gestellten Anforderungen erfüllt. Im Gegensatz zum Systemtest erfolgt der Abnahmetest nicht mehr im Verantwortungsbereich des Herstellers. Der Abnahmetest wird in der realen Betriebsumgebung beim Kunden bzw. Auftraggeber durchgeführt. Der Test erfolgt somit, falls dies nicht bereits im Systemtest erfolgte, mit authentischen Eingabedaten [Hof08].

### 4.6.3 Simulations- und Testschnittstelle

Der Demonstrator bietet über eine Simulations- und Testschnittstelle die Möglichkeit, Testfälle automatisiert ablaufen und auswerten zu können. In Abbildung 4.44 ist der Aufbau schematisch dargestellt. Durch diese Schnittstelle kann das Steuergerät an ein Testsystem gekoppelt werden, durch das zum einen eine Stimulation der Sensorik und zum anderen eine Überwachung der Aktorik möglich ist.

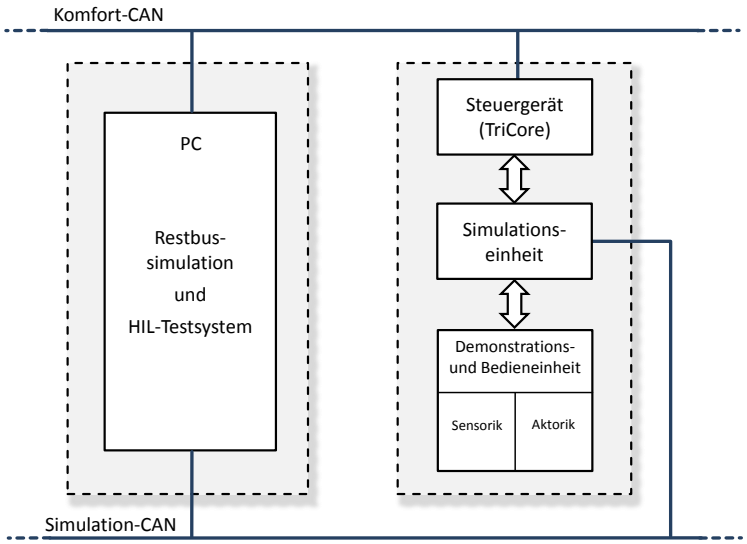


**Abbildung 4.44:** Schematische Darstellung der Simulations- und Testschnittstelle

Die Kommunikation zwischen den einzelnen Steuergeräten (vgl. Abbildung 3.2, Kapitel 3) erfolgt für das im Rahmen dieser Fallstudie behandelte PKW-Komfortsystem über den Komfort-CAN. Während des regulären Betriebes ist die Simulationseinheit für die Kommunikation zwischen Steuergerät und Demonstrations- und Bedieneinheit transparent. Darüber hinaus wurde durch die Simulationseinheit eine Schutzfunktion der Aktorik der Demonstrations- und Bedieneinheit vor unzulässigen Betriebszuständen realisiert.

Für den Testbetrieb kann die Demonstrations- und Bedieneinheit durch die Simulationseinheit deaktiviert werden, die anschließend diese Komponente vollständig simuliert. Weiterhin erfolgt über den Simulation-CAN die Kommunikation zwischen der Simulationseinheit und dem zentralen HIL-Testsystem, so dass durch diese Kopplung ein Hardware-in-the-Loop(HIL)-System entsteht, um auf die Ausgangssignale des Steuergerätes entsprechend reagieren zu können. Dadurch kann die Aktorik gezielt stimuliert und automatisiert getestet werden. Der Einsatz von HIL in der Fallstudie spiegelt den aktuellen Stand der Technik der systematischen Prüfung von Steuergeräten wider. Besonders im Automobilbereich ist das HIL-Testing bei eingebetteten Systemen notwendig, um komplexe Systeme testen zu können. Durch eine Restbussimulation wird über den Komfort-CAN die Kommunikation mit anderen simulierten nicht physikalisch vorhandenen Fahrzeugsystemen ermöglicht.

Eine detaillierte Beschreibung der Hardware und Simulationseinheit des Systems findet sich in [Mic09].

### 4.6.4 Automatisiertes Testen in der Fallstudie

Für die Fallstudie wurde ein automatisiertes Testsystem auf Basis des Unittestframeworks JUnit entwickelt. JUnit[4] ist ein open-source Framework zur Automatisierung von Unittests für JAVA, es kann aber auch zum Testen eines Modulverbunds oder des Gesamtsystems eingesetzt werden. Der Grundgedanke der Unittest-Frameworks, die auf zahlreiche Programmiersprachen portiert wurden, ist die inkrementelle Definition von Testfällen parallel zur Entwicklung der Software [TLG06].

Das Ergebnis eines JUnit-Tests kann prinzipiell nur entweder das erfolgreiche Bestehen oder das Misslingen des durchgeführten Testfalls sein. Falls der Testfall scheitert, wird der Grund als Error oder Failure gekennzeichnet.

Mit den Methoden, die die JUnit-Klasse *„Assert“* bereitstellt, können Bedingungen definiert werden, die über den Ausgang eines Testfalls entscheiden. Wird diese Bedingung verletzt oder nicht erfüllt, führt dies zu einem protokollierten Fehler. Die Definition dieser Bedingungen kann direkt aus der Spezifikation bzw. den Anforderungen erfolgen.

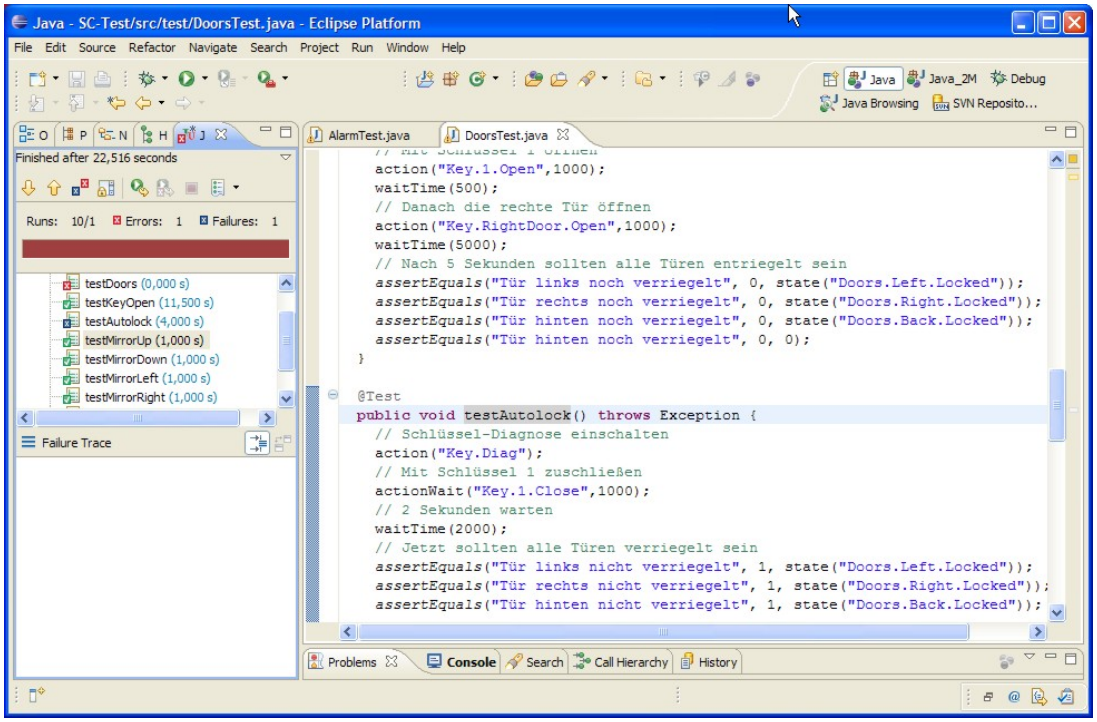


**Abbildung 4.45:** Automatisiertes Testen auf Basis von JUnit

[4] www.junit.org

Die automatisierten Tests in der Fallstudie beschränken sich im Wesentlichen auf Akzeptanz- bzw. Integrationstests entsprechend des modifizierten V-Modells (siehe Abschnitt 3.3). Die Testfälle wurden dabei als rein funktionsorientierte Tests definiert, so dass die innere Struktur unberücksichtigt bleibt. Dies bewahrt die Unabhängigkeit der definierten Testfälle von der eigentlichen Implementierung. Eine Prüfung erfolgt somit nur gegen die festgelegten Anforderungen an das gesamte System. Zu jeder Anforderung existiert mindestens ein Testfall, der soweit dies möglich ist, automatisiert durchgeführt und bewertet wird.

Vor dem Ausführen eines Testfalls wird das System durch die Simulationseinheit in den entsprechenden Modus versetzt. Anschließend kann das Testsystem über die Simulations- und Testschnittstelle mit dem Steuergerät interagieren und mit der Durchführung der Testfälle beginnen.

In Abbildung 4.45 ist exemplarisch ein Auszug aus der Testfallimplementierung für das Zentralverriegelungssteuergerät dargestellt. Durch den Testfall *„testAutoLock“* wird zum Beispiel die Anforderungen *„Auto-Lock“* mit der ID 100010 (siehe [Gar09]) geprüft. Zum Bestehen dieses Testfalls sind wie abgebildet diverse Bedingungen zu erfüllen.

Nach erfolgreicher Durchführung aller definierten Testfälle ist die Implementierung im Sinne der Anforderungen als korrekt zu bewerten. Durch das eingesetzte Testsystem konnte eine effiziente und wirksame Überprüfung der gestellten Anforderungen automatisiert durchgeführt werden.

# 5 Zusammenfassung

Es wurde eine umfangreiche Fallstudie eines modellbasierten Entwicklungsprozesses von der Anforderungsbeschreibung bis hin zur Implementierung, Integration und Test von Steuergeräten durchgeführt. Neben einigen Grundlagen zum AUTOSAR-Standard und zur Architekturevaluation wurden hier zunächst der entstandene Demonstrator, ein vereinfachtes PKW-Komfortsystem bestehend aus vier über CAN vernetzten Steuergeräten, vorgestellt. Weiterhin wurden die einzelnen Entwicklungsschritte zusammen mit den verwendeten Tools (DOORS, SystemDesk, Matlab/Simulink/Stateflow, TresosECU) beschrieben und ausgewählte Implementierungsdetails und Workarounds diskutiert. Ein weiterer Teil ist der Konfiguration der AUTOSAR-Basis-Software gewidmet. Neben der Umsetzung wurde eine Methode zur Architekturevaluation von AUTOSAR-konform beschrieben Systemarchitekturen vorgestellt. Abschließend sollen nun einige Beobachtungen bei der Umsetzung der Fallstudie diskutiert werden, insbesondere in wie weit die Konzepte von AUTOSAR in den verfügbaren Tools bereits umgesetzt wurden und die Integration in die Toolkette harmonierte.

## 5.1 Diskussion der Toolkette

**AUTOSAR – Theorie und Praxis / Entwicklungsstand**

Bei der Betrachtung des gesamten Entwicklungsprozesses ist festzustellen, dass zwar für jede Phase Entwicklungswerkzeuge mit einer Basisfunktionalität vorhanden waren, es jedoch an einem reibungslosen Datenaustausch zwischen den Phasen mangelte. Sowohl mit Matlab/Simulink/Stateflow in Kombination mit automatischer Codegenerierung als auch mit direkter Implementierung in C konnte eine stabile Software auf AUTOSAR-Basis für einzelne Steuergeräte erstellt werden. Eine durchgehende Unterstützung für aus mehreren Steuergeräten bestehenden Systemen war nicht verfügbar.

In der AUTOSAR-Methodologie ist für die Entwicklung eine Orientierung an Funktionen statt an Steuergeräten vorgesehen. Die vom Entwickler erstellten Softwarekomponenten sollen durch Unterstützung von Tools optimal auf einzelne Steuergeräte eines Verbundes verteilt werden können (s. Abschnitt 2.3). Die automatisierte Verteilung von Softwarekomponenten auf Steuergeräte wurde noch nicht unterstützt. Das Konzept, miteinander kommunizierenden Softwarekomponenten auch auf unterschiedlichen Steuergeräten beliebig verteilen zu können,

lässt im Prinzip eine automatisierte Kommunikationsstruktur sinnvoll erscheinen. Tools sollten die Verbindungswege für den Nachrichtenaustausch über RTE und Basismodule weitgehend eigenständig konfigurieren, um die Entwickler davon zu entlasten. Zum Zeitpunkt der Fallstudie war das Verlagern einer Softwarekomponente von einem Steuergerät auf ein anderes mit einem enormen Konfigurationsaufwand für die Nachrichtensignale in der Basisebene verbunden.

Das Konzept von AUTOSAR, Softwarekomponenten unabhängig von der Ziel-Hardware verwenden zu können, wurde in den Entwicklungswerkzeugen umgesetzt. Die Tools basieren diesbezüglich auf den Vorgaben der AUTOSAR-Spezifikation und erzeugen funktionstüchtige Resultate. Die eingesetzte Tool-Kette (siehe Kapitel 4) zeigte jedoch Lücken und Inkompatibilitäten beim Datenaustausch zwischen Entwicklungswerkzeugen von unterschiedlichen Herstellern. Auf fast jeder Ebene der Tool-Kette kam ein Software-Paket eines anderen Hersteller zum Einsatz. Ein direkter Datenaustausch war auf den oberen Ebenen nicht vorhanden. Von DOORS und Systemdesk aus konnten keine Dateien in MATLAB/Simulink eingebunden werden. Die von TargetLink erzeugten Dateien konnten nach einer einfachen Modifikation der Pfadnamen von I/O-Signalen vom nachfolgenden Tool tresosECU erfolgreich eingebunden werden. Einen erheblich höheren Aufwand stellte das Anfertigen fehlender Systembeschreibungsdateien dar. Eine Datei mit der sogenannten Top Level Composition wird benötigt, aber keines der verfügbaren Werkzeuge konnte sie automatisch erzeugen oder das Erstellen unterstützen. Dieses Defizit musste mit selbst entwickelter Software in Form von Shell-Skripten behoben werden.

Der Datenfluss in der Tool-Kette wies auf der unteren Ebene keine größeren Probleme auf, d.h. die von tresosECU generierten Quellcode- und Make-Dateien konnten im Normalfall vom C-Compiler verarbeitet werden. Unter bestimmten Umständen war es jedoch erforderlich, alle generierten Dateien zu löschen und vollständig neu erzeugen zu lassen.

In den nächsten Absätzen erfolgt eine Bewertung des Entwicklungsstandes der Software-Pakete MATLAB/Simulink/Stateflow mit TargetLink-Erweiterung und tresosECU.

### Anmerkungen zu MATLAB/Simulink/Stateflow und TargetLink

Das aus MATLAB/Simulink/Stateflow und TargetLink zusammengesetzte Software-Paket bot eine Grundfunktionalität zum Erstellen von modellbasierten AUTOSAR-Softwarekomponenten. Allerdings waren noch Fehler enthalten und wichtige Elemente fehlten. Eine gravierende Unzulänglichkeit lag in der Generierung von C-Code durch TargetLink. Die fehlende Auswertung des Rückgabewertes der RTE-Funktion zum Nachrichtenempfang führte zu Zufallswerten. Die Problemlösung war mit einem verhältnismäßig hohen Aufwand für jede zu emp-

fangende Nachricht verbunden. Unter Berücksichtigung dieser Maßnahmen wurden von TargetLink stabile Softwarekomponenten aus den MATLAB/Simulink-Modellen generiert. Das Simulieren des Betriebsverhaltens der Softwarekomponenten war möglich und stellt eine wichtige Unterstützung im Entwicklungsprozess dar.

Ein großer Mangel des Software-Paketes war das Fehlen von Timern. Für die Messung von Zeitabständen oder das Auslösen von Timeouts wurden Timer in vielen Anwendungen benötigt. Der Mangel konnte durch selbst erstellte Taktgeneratoren mit angekoppelten Zählern ausgeglichen werden. Auch dies führte zu einem deutlich höheren Entwicklungsaufwand.

Eine strittige Frage war, ob eine Generierung von Compositions-Dateien durch TargetLink vorgesehen werden sollte. Beim verwendeten Entwicklungsstand hätten zumindest alle Softwarekomponenten eines Steuergerätes in einem MATLAB/Simulink-Projekt aufgenommen werden können, aus dem eine Composition hätte automatisch erzeugt werden können.

**Anmerkungen zu tresosECU**

Das Software-Paket tresosECU stellte die AUTOSAR-Basismodule und -RTE zur Verfügung. Die Basis-Software konnte über die Bedienoberfläche konfiguriert werden. Zusammen mit externen Softwarekomponenten generierte tresosECU AUTOSAR-konformen Code für Steuergeräte.

Eine große Schwachstelle war die bisherige Lösung für die I/O Hardware Abstraction. Das entsprechende Modul musste vom Entwickler selbst entwickelt, konfiguriert und gewartet werden. Da es sich um eine elementare Systemkomponente handelt und da sie einen verhältnismäßig hohen Arbeitsaufwand erfordert, wiegte dieses Defizit besonders schwer.

Die Konfiguration der Module erforderte einen sehr hohen Zeitaufwand und besteht insbesondere bei Nachrichtensignalen aus einer Unzahl von gleichförmigen Arbeitsabläufen. Hier wäre der Einsatz von Makros sehr empfehlenswert gewesen. Zum Zeitpunkt der Fallstudie konnten neue Projekte nur mit vollständig unkonfigurierten Modulen angelegt werden. Dadurch war zum einen ein hoher zeitlicher Aufwand für Basiseinstellungen und zum anderen eine sehr umfassende Einarbeitungszeit erforderlich. Vorkonfigurierte Module wären hier eine sinnvolle Abhilfe gewesen.

Aus der Konfigurationsoberfläche ließen sich Generierungsläufe starten, deren Fehlerausgaben in vielen Fällen nicht zur Lokalisierung der Probleme beigetragen hatten. In der verwendeten Version waren viele Arbeitsschritte über eine Kommandozeilenumgebung abzuwickeln und das Editieren von Projektdateien unumgänglich. Vereinfachungen und eine Steigerung des Arbeitskomforts hätten erheblich zur Fehlerfreiheit und Effizienz von Projekten beitragen können.

### Anmerkungen zur Umsetzung

Der AUTOSAR-Demonstrator konnte mit einigen Komplikationen und Einschränkungen realisiert werden. Die zum Zeitpunkt der Umsetzung noch nicht einsetzbare Software zur Definition der Systemarchitektur auf der Ebene von Softwarekomponenten und RTE stellte ein wesentliches Defizit bei der Entwicklung dar. Die notwendigen Arbeiten mussten von Hand und mit Hilfe von Shell-Skript-Programmierung ausgeführt werden. Auch in Bezug auf Systeme, die aus mehreren Steuergeräten bestehen, fehlte eine Unterstützung. Für einen der zentralen Ansätze von AUTOSAR, die effiziente Verteilung von Softwarekomponenten auf die Steuergeräte im Systemverbund zu automatisieren, waren keine Softwaretools verfügbar.

Die in Kernbereichen angesiedelten Werkzeuge TargetLink von dSPACE und tresosECU von Elektrobit wiesen in den verwendeten ersten Release-Versionen noch Fehler und Implementierungsschwächen auf, die sich größtenteils umgehen oder ausgleichen ließen. Insbesondere war ein reibungsloser Datenaustausch zwischen Werkzeugen unterschiedlicher Hersteller an zentralen Punkten nicht gegeben.

Eine effiziente Entwicklungsarbeit konnten die Werkzeuge in mehreren Bereichen nicht gewährleisten. Einige Schritte waren nur umständlich und zeitintensiv durchzuführen. Die Adaption von bereits vorhandenen MATLAB/Simulink-Modellen auf die AUTOSAR-Plattform war mit einem relativ hohen Aufwand verbunden.

Unter Anwendung der hier dokumentierten Maßnahmen konnten mit den teilweise frühen Versionen der Software-Pakete letztlich stabil funktionierende Steuergeräte auf AUTOSAR-Basis programmiert werden.

### Notwendige Entwicklungen der Tools

Für eine durchgängige Entwicklung von AUTOSAR-Systemen werden Software-Pakete für die Erstellung der Systemarchitektur benötigt, die insbesondere Definitionsdaten von Werkzeugen anderer Anbieter verarbeiten. Für einen reibungslosen Datentransfer zwischen verschiedenen Tools sind von allen beteiligten Herstellern Absprachen und eventuell strengere Auslegungen des bestehenden AUTOSAR-Standards erforderlich. Eine besondere Herausforderung wird die Entwicklung der Tools für die automatisierte Verteilung von Softwarekomponenten darstellen.

Die für die Fallstudie verfügbaren Software-Pakete wie TargetLink und tresosECU müssen an mehreren Stellen verbessert und erweitert werden. Von einer Behebung der Fehler abgesehen, sollte vor allem die Effizienz der Entwicklung durch Automatisierungen und die Vermeidung von nahezu redundanten Eingaben gesteigert werden. Im Speziellen werden in TargetLink Timer benötigt und in

tresosECU sollte vordringlich sowohl eine bessere Lösung für die I/O Hardware Abstraction als auch eine sinnvolle Vorkonfiguration der Basismodule angeboten werden.

Ein besonderer Anspruch an mehrere Tool-Hersteller wäre die Automatisierung der Kommunikationsstrukturen zwischen Softwarekomponenten innerhalb eines Steuergerätes und möglichst auch im Systemverbund. Schwierigkeiten könnten sich zwar im Konflikt mit einer bestehenden Kommunikationsmatrix im Fahrzeug ergeben, sie wären aber prinzipiell lösbar, beispielsweise durch entsprechende Vorgaben oder getrennte Netze. Ein weitestgehender Wegfall der Entwicklungsarbeit an den Kommunikationswegen würde einerseits zu einer enormen Zeitersparnis führen und anderseits die Sicherheit und Zuverlässigkeit erhöhen.

Die aufgeführten Probleme können nicht als fundamentale Schwachstellen angesehen werden und stellen die guten Zukunftsaussichten von AUTOSAR nicht in Frage. AUTOSAR zeigt durchaus das Potenzial die ansteigende Komplexität von elektronischen Systemen in Kraftfahrzeugen beherrschbar zu machen.

## 5.2 Potential und Ausblick

Die Bedeutung, die zukünftig dem AUTOSAR-Standard in der Entwicklung von Steuergeräte-Software zukommen wird, wird es zwingend erforderlich machen, Maßnahmen und Ansätze zur konstruktiven und analytischen Qualitätssicherung und -Optimierung in die Entwurfsmethodik zu integrieren. In diesem Bericht wurde bereits eine Vielzahl von entsprechenden Ansatzpunkten aufgezeigt.

Die modellbasierten, modularen und durchgängig werkzeuggestützten Entwicklungskonzepte von AUTOSAR können, wie anhand der Fallstudie dargelegt wurde, bereits maßgeblich zur Beherrschbarkeit der Komplexität derartiger Systeme bis zur vollständigen Implementierung beitragen.

Die Fokussierung von AUTOSAR auf einen konsequent Architektur-basierten Entwurf ist dabei prädestiniert für frühzeitige Ansätze zur Qualitätsabschätzung durch Architektur-Evaluation, wie sie in diesem Bericht skizziert wurden. Aufbauend auf dem beschriebenen Framework kann in zukünftigen Arbeiten der Ansatz schrittweise verfeinert werden:

1. Durch inkrementelle Weiterentwicklungen und wiederholte Validierung der Bewertungsmetriken in Hinblick auf die Artefakte des AUTOSAR-Metamodells und der resultierenden Implementierungen durch die in nachfolgenden Phasen eingesetzten Authoring Tools kann die Aussagekraft und Verlässlichkeit der Bewertungsergebnisse weiter erhöht werden.

2. Anhand der Bewertungsstruktur können durch Tracing Rückschlüsse auf die das Bewertungsergebnis maßgeblich beeinflussende Architektur-Arte-

fakte getroffen werden, um auf diese Weise Optimierungspotentiale aufzuzeigen.

3. Neben der automatisierten Auswertung von Architektur-Bewertungen können auch anschließende Optimierungsschritte zur Verbesserung des Entwurfs durch entsprechende Hilfsfunktionen und Werkzeuge (z.B. zum Refactoring) unterstützt werden.

Bezogen auf den ersten Punkt können auch die Neuerungen in der kommenden Version 4 des AUTOSAR-Standards von großem Interesse sein. Die weiter gestiegene Detaillierung des Systementwurfs, insbesondere in der Phase der Architektur-Spezifikation, schafft neue Möglichkeiten für die Architektur-Evaluation. Die Neuerungen werden auch Einfluss auf die Durchgängigkeit bestehender Werkzeugverbünde haben, wie sie in diesem Bericht exemplarisch beschrieben wurden.

Neben der Qualitätssicherung für nicht-funktionale Anforderungen durch Architektur-Evaluation wurde die Überprüfung der funktional korrekten Umsetzung der Anforderungen durch Testansätze begleitend zu den Entwurfsphasen gemäß dem rechten Ast des V-Modells beschrieben. Die beschriebene exemplarische Umsetzung von automatisierten Tests zur direkten Überprüfung von Szenarien aus der Anforderungsspezifikation können im nächsten Schritt kombiniert werden mit Ansätzen, die weitere Ebenen des V-Modells abdecken. Dazu zählen neben modellbasierten Testverfahren zur Überprüfung des technischen Designs und Architektur-basierten Ansätzen für Komponententests auch auf Simulationen basierende Validierungstechniken für das System, die frühzeitig und ohne konkrete Hardware das zu erwartende Systemverhalten nachbilden. Beispielsweise bietet SystemDesk von dSpace die Möglichkeit, wesentliche Abläufe des Virtual Function Bus zu simulieren und so die Interaktion zwischen verschiedenen Steuergeräten auch ohne bereits erfolgte Verteilung der Komponenten im Netzwerk zu überprüfen.

## 5.3 Acknowledgement

Wir bedanken uns herzlich für die großzügige Unterstützung durch die Volkswagen AG, Elektrobit Automotive GmbH, Telelogic Deutschland GmbH, The MathWorks GmbH und dSPACE GmbH. Weiterhin bedanken wir uns bei allen an der Fallstudie unterstützenden und mitwirkenden Institute und Personen,

- Institut für Programmierung und Reaktive Systeme: Michaela Huhn
- Institut für Software Systems Engineering: Bernhard Rumpe, Christian Basarke, Alexej Beresnev
- Institut für Verkehrssicherheit und Automatisierungstechnik: Eckehard Schnieder, Matthias Hübner
- Institut für Regelungstechnik: Markus Maurer, Sebastian Ohl, Tobias Michaels

alle an der Technischen Universität Braunschweig, ohne die eine Fallstudie in diesem Umfang nicht möglich gewesen wäre.

# Literaturverzeichnis


[AUT06] AUTOSAR Administration: AUTOSAR Methodology. Version: 2006. http://www.autosar.org/download/AUTOSAR_Methodology.pdf. 2006. – Forschungsbericht 1, 6, 7, 48

[AUT07a] AUTOSAR GbR: *AUTOMOTIVE OPEN SYSTEM ARCHITECTURE*. Website. http://www.autosar.org. Version: 2007 3, 8, 9, 16, 40

[AUT07b] AUTOSAR GbR: *AUTOSAR Layered Software Architecture*. Website. http://www.autosar.org/download/AUTOSAR_LayeredSoftwareArchitecture.pdf. Version: 2007 30, 32, 33, 35

[AUT07c] AUTOSAR GbR: *AUTOSAR Software Component Template*. Website. http://www.autosar.org/download/AUTOSAR_SoftwareComponentTemplate.pdf. Version: 2007 108

[AUT07d] AUTOSAR GbR: *AUTOSAR Specification of ICU Driver*. Website. http://www.autosar.org/download/AUTOSAR_SWS_ICU_Driver.pdf. Version: 2007 37

[AUT07e] AUTOSAR GbR: *Specification of ADC Driver*. Website. http://www.autosar.org/download/AUTOSAR_SWS_ADC_Driver.pdf. Version: 2007 37

[AUT07f] AUTOSAR GbR: *Specification of CAN Driver*. Website. http://www.autosar.org/download/AUTOSAR_SWS_CAN_Driver.pdf. Version: 2007 36

[AUT07g] AUTOSAR GbR: *Specification of CAN Interface*. Website. http://www.autosar.org/download/AUTOSAR_SWS_CAN_Interface.pdf. Version: 2007 35

[AUT07h] AUTOSAR GbR: *Specification of CAN Transport Layer*. Website. http://www.autosar.org/download/AUTOSAR_SWS_CAN_TP.pdf. Version: 2007 35

[AUT07i] AUTOSAR GbR: *Specification of Communication*. Website. http://www.autosar.org/download/AUTOSAR_SWS_COM.pdf. Version: 2007 34

[AUT07j] AUTOSAR GbR: *Specification of DIO Driver*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_DIO_Driver.pdf`. Version: 2007 37

[AUT07k] AUTOSAR GbR: *Specification of I/O Hardware Abstraction*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_IO_HWAbstraction.pdf`. Version: 2007 36, 117

[AUT07l] AUTOSAR GbR: *Specification of PDU Router*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_PDU_Router.pdf`. Version: 2007 34

[AUT07m] AUTOSAR GbR: *Specification of PORT Driver*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_Port_Driver.pdf`. Version: 2007 37

[AUT07n] AUTOSAR GbR: *Specification of PWM Driver*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_PWM_Driver.pdf`. Version: 2007 37

[AUT07o] AUTOSAR GbR: *Specification of RTE*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_RTE.pdf`. Version: 2007 91

[AUT07p] AUTOSAR GbR: *Specification of GPT Driver*. Website. `http://www.autosar.org/download/AUTOSAR_SWS_GPT_Driver.pdf`. Version: März 2007 37

[AUT09a] *AUTOSAR Übersicht*. Website. `http://www.elektroniknet.de/home/automotive/autosar/`. Version: 2009 8, 9

[AUT09b] *AUTOSAR Technologie*. Website. `http://www.dspace.de/ww/de/gmb/home/products/our_solutions_for/autosaratdspace.cfm`. Version: 2009 38

[Bal98] Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Heidelberg Berlin : Spektrum Akademischer Verlag, 1998. – ISBN 3–8274–0065–1 3, 47, 48

[BCK03] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software Architecture in Practice*. 2. Auflage. Pearson, 2003 3, 10, 66, 68, 76

[CFGK05] Conrad, M. ; Fey, I. ; Grochtmann, M. ; Klein, T.: Modellbasierte Entwicklung eingebetteter Fahrzeugsoftware bei Daimler-Chrysler. In: *Informatik-Forschung und Entwicklung* 20 (2005), Nr. 1, S. 3–10 97

[CN07] Clements, Paul ; Northrop, Linda: *Software Product Lines : Practices and Patterns*. 6. Auflage. Addison Wesley, 2007 74

[Dau03] Daum, Udo Berthold; M. Berthold; Merten: *System Architecture With XML*. Elsevier, 2003. – ISBN 9781558607453 108

[dSP06] dSPACE: *AUTOSAR Modeling Guide (TargetLink 2.2)*, 2006 103, 106

[Ele07] Elektrobit Automotive GmbH: *tresos Studio 2007.b : Developers Guide*, 2007 125

[FGB+07] Florentz, B. ; Goltz, U. ; Braam, J. ; Ernst, R. ; Saul, T.: Architekturevaluation: Qualitätssicherung in frühen Entwicklungsphasen. In: *Achtes Symposium AAET 2007 - Automatisierungs-, Assistenzsysteme und eingebettete Systeme für Transportmittel*, 2007, S. 238–248 3, 68, 70, 72, 76, 83

[Gar09] Garbers, Henning: *Aufbau und Implementierung einer AutoSar-Plattform auf Basis eines TriCore-Controllers*, TU Braunschweig, Diplomarbeit, 2009 3, 41, 53, 133

[Hof08] Hoffmann, D.: *Software-Qualität*. Heidelberg Berlin : Springer-Verlag, 2008 129, 130

[Hor05] Horstmann, M.: *Verflechtung von Test und Entwurf für eine verlässliche Entwicklung eingebetteter Systeme im Automobilbereich*, Technische Universität Braunschweig, Institut für Verkehrssicherheit und Automatisierungstechnik, Diss., 2005 97

[IAB09] *Das V-Modell*. Website. `http://www.v-modell.iabg.de/`. Version: 2009 47

[IEE90] IEEE Standards Board: Standard Glossary of Software Engineering Terminology / Standards Coordinating Committee of the IEEE Computer Society. 1990. – Forschungsbericht. – ISBN ISBN 1–55937–067–X 126

[Inf05] Infineon Technologies AG: *TriCore TC1766 Data Sheet*, 2005 115

[KF08] Kindel, Olaf ; Friedrich, Mario: *Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis*. dpunkt Verlag, 2008. – 292 S. – ISBN 3898645630 1, 3, 39, 40

[KMS08] Kessler, M. ; Müller, O. ; Schäfer, G.: *C++ in der Automotive-Software-Entwicklung*. Website. `http://www.elektroniknet.de/index.php?id=2147`. Version: 2008 94

[Leh04] Lehmann, E.: *Time Partition Testing - Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen*, Technische Universität Berlin, Diss., 2004 129

[Lig02] Liggesmeyer, Peter: *Software-Qualität. Testen, Analysieren und Verifizieren von Software*. Heidelberg Berlin : Spektrum Akademischer Verlag, 2002 3, 126, 128

[Lig05] Liggesmeyer, Peter ; Rombach, Dieter (Hrsg.): *Software-Engineering eingebetteter Systeme: Grundlagen-Methodik-Anwendungen*. Heidelberg Berlin : Spektrum Akademischer Verlag, 2005 126

[MHH+03] Mutz, M. ; Harms, M. ; Horstmann, M. ; Huhn, M. ; Bikker, G. ; Krömke, C. ; Lange, K. ; Goltz, U. ; Schnieder, E. ; Varchmin, J.-U.: Ein durchgehender modellbasierter Entwicklungsprozess für elektronische Systeme im Automobil. In: Verkehrstechnik, VDI Gesellschaft F. (Hrsg.) ; VDI Tagung Elektronik im Kraftfahrzeug (Veranst.): *Elektronik im Kraftfahrzeug* VDI Tagung Elektronik im Kraftfahrzeug, VDI Verlag GmbH, September 2003, S. 43–76. – ISBN: 3-18-091789-X 97

[Mic09] Michaels, Tobias: *Entwicklung und Aufbau eines vernetzten Automotive Demonstrators*, TU Braunschweig, Diplomarbeit, 2009 3, 41, 53, 131

[Mye89] Myers, G. J.: *Methodisches Testen von Programmen*. München Wien : R. Oldenbourg Verlag, 1989 126

[Poh08] Pohl, Klaus: *Requirements Engineering: Grundlagen, Prinzipien,Techniken*. 2. Auflage. dpunkt.Verlag, 2008 3, 79

[SMWM09] Saust, Falko ; Müller, Tobias C. ; Wille, Jörn M. ; Maurer, Markus: Entwicklungsbegleitendes Simulations- und Testkonzept für autonome Fahrzeuge in städtischen Umgebungen. In: *AAET 2009. Automatisierungs-, Assistenzsysteme und eingebettete Systeme für Transportmittel*, 2009 129

[SZ06] Schäuffele, Jörg ; Zurawka, Thomas: *Automotive Software Engineering*. Vieweg, 2006. – ISBN 3–8348–0051–1 1, 3, 12, 53

[Tel] Telelogic AB: *DOORS* 53

[TLG06] Thiruvathukal, G.K. ; Laeufer, K. ; Gonzalez, B.: Unit Testing Considered Useful. In: *IEEE Computational Science and Engineering* 8 (2006), S. 76–87 132

[Wor07a] World Wide Web Consortium (W3C): *XML Path Language (XPath) 2.0*. Website. `http://www.w3.org/TR/xpath20/`. Version: 2007 123

[Wor07b] World Wide Web Consortium (W3C): *XQuery 1.0 and XPath 2.0 Data Model (XDM)*. Website. `http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/`. Version: 2007 123

Technische Universität Braunschweig
Informatik-Berichte ab Nr. 2007-01

| | | |
|---|---|---|
| 2007-01 | M. Conrad, H. Giese, B. Rumpe, B. Schätz (Hrsg.) | Tagungsband Dagstuhl-Workshops MBEES: Modellbasierte Entwicklung eingebetteter Systeme III |
| 2007-02 | J. Rang | Design of DIRK schemes for solving the Navier-Stokes-equations |
| 2007-03 | B. Bügling, M. Krosche | Coupling the CTL and MATLAB |
| 2007-04 | C. Knieke, M. Huhn | Executable Requirements Specification: An Extension for UML 2 Activity Diagrams |
| 2008-01 | T. Klein, B. Rumpe (Hrsg.) | Workshop Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen, Tagungsband |
| 2008-02 | H. Giese, M. Huhn, U. Nickel, B. Schätz (Hrsg.) | Tagungsband des Dagstuhl-Workshopss MBEES: Modellbasierte Entwicklung eingebetteter Systeme IV |
| 2008-03 | R. van Glabbeek, U. Goltz, J.-W. Schicke | Symmetric and Asymmetric Asynchronous Interaction |
| 2008-04 | R. van Glabbeek, U. Goltz, J.-W. Schicke | On Synchronous and Asynchronous Interaction in Distributed Systems |
| 2008-05 | M. V. Cengarle, H. Grönniger B. Rumpe | System Model Semantics of Class Diagrams |
| 2008-06 | M. Broy, M. V. Cengarle, H. Grönniger B. Rumpe | Modular Description of a Comprehensive Semantics Model for the UML (Version 2.0) |
| 2008-07 | C. Basarke, C. Berger, K. Berger, K. Cornelsen, M. Doering J. Effertz, T. Form, T. Gülke, F. Graefe, P. Hecker, K. Homeier F. Klose, C. Lipski, M. Magnor, J. Morgenroth, T. Nothdurft, S. Ohl, F. Rauskolb, B. Rumpe, W. Schumacher, J. Wille, L. Wolf | 2007 DARPA Urban Challenge Team CarOLO - Technical Paper |
| 2008-08 | B. Rosic | A Review of the Computational Stochastic Elastoplasticity |
| 2008-09 | B. N. Khoromskij, A. Litvinenko, H. G. Matthies | Application of Hierarchical Matrices for Computing the Karhunen-Loeve Expansion |
| 2008-10 | M. V. Cengarle, H. Grönniger B. Rumpe | System Model Semantics of Statecharts |
| 2009-01 | H. Giese, M. Huhn, U. Nickel, B. Schätz (Herausgeber) | Tagungsband des Dagstuhl-Workshops MBEES: Modellbasierte Entwicklung eingebetteter Systeme V |
| 2009-02 | D. Jürgens | Survey on Software Engineering for Scientific Applications: Reuseable Software, Grid Computing and Application |
| 2009-03 | O. Pajonk | Overview of System Identification with Focus on Inverse Modeling |
| 2009-04 | B. Sun, M. Lochau, P. Huhn, U. Goltz | Parameter Optimization of an Engine Control Unit using Genetic Algorithms |
| 2009-05 | A. Rausch, U. Goltz, G. Engels, M. Goedicke, R. Reussner | LaZuSo 2009: 1. Workshop f̈ur langlebige und zukunftsfähige Softwaresysteme 2009 |
| 2009-06 | T. C. Müller, M. Lochau, S. Detering, F. Saust, H. Garbers, L. Märtin, T. Form, U. Goltz | Umsetzung eines modellbasierten durchgängigen Enwicklungsprozesses für AUTOSAR-Systeme mit integrierter Qualitätssicherung |